te КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)

Кафедра кібербезпеки та комп'ютерної інженерії
(назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

на тему:

Система батьківського контролю для веб-сервісів

Хоменко Влад Романович
(прізвище, ім'я та по батькові здобувача повністю)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій

(факультет)

Кафедра кібербезпеки та комп'ютерної інженерії

(назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
к.т.н., доцент Максим ДЕЛЕМБОВСЬКИЙ

_____
„___" _____20___ року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

_Система батьківського контролю для веб-сервісів_____

(назва)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Хоменко Влад Романович

(прізвище, ім'я та по батькові повністю)

_125_«Кібербезпека та захист інформації»__

(спеціальність)

безпека інформаційних і комунікаційних систем_____

(освітня програма)

Група БІКСм-24_____

Керівник_Гуменний Д. О._____

(прізвище та ініціали)

к.т.н., доц._____

(вчене звання, науковий ступінь)

Рецензент_Рябчун Ю.В._____

(прізвище та ініціали)

Ідентичність підтверджую

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Кафедра: кібербезпеки та комп'ютерна інженерія
Освітній рівень: магістр
Спеціальність: 125 «Кібербезпека та захист інформації»
ОПП: Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
к.т.н., доцент Максим ДЕЛЕМБОВСЬКИЙ

_____
„___"_____20___ року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

Хоменко Влад Романович
(прізвище, ім'я та по батькові здобувача)

1. Тема роботи Система батьківського контролю для веб-сервісів
затверджена наказом ректора КНУБА № 1635/23.2/25 від 30.09.2025
2. Керівник роботи Гуменний Дмитро Олександрович, к.т.н, доцент кафедри кібербезпеки та комп'ютерної інженерії
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)
3. Термін подання здобувачем роботи до захисту 08.12.2025
4. Зміст пояснювальної записки за розділами:
Р.1. Аналіз предметної області та постановка задачі
Р.2. Архітектура системи
Р.3. Проєктування бази даних
Р.4. Практична реалізація та тестовий приклад
Р.5. Висновки
5. Графічний матеріал за розділами:
Р. 1. Ключові аспекти системи
Р. 2. Постановка задачі
Р. 3. Концептуальна схема системи
Р. 4. Функціональна блок-схема потоку
Р. 5. Діаграма сутностей-відносин
Р. 6. Практична реалізація та результат розробки

6. Консультанти розділів кваліфікаційної випускної роботи

Розділи	Прізвища, ініціали та посади консультанта	Перевірив	
		дата	підпис
Розділ 1.			
Розділ 2.			
Розділ 3.			
Розділ 4.			

7. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1.	Червень 2025 р.
Розділ 2.	Вересень 2025 р.
Розділ 3.	Жовтень 2025 р.
Розділ 4.	Листопад 2025 р.
Остаточне оформлення роботи	Листопад 2025 р.
Направлення роботи на рецензування, перевірку на плагіат	Листопад 2025 р.
Попередній захист роботи на кафедрі	Грудень 2025 р.

8. Дата видачі завдання _26.06.2024_____

Керівник _____ _Гуменний Д.О._
(підпис) (прізвище та ініціали)

Здобувач _____ _Хоменко В.Р.___
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Хоменко В. Р. Система батьківського контролю для веб-сервісів.

Кваліфікаційна робота магістра за спеціальністю: 125 «Кібербезпека та захист інформації», спеціалізація: «Безпека інформаційних і комунікаційних систем». – Київський національний університет будівництва та архітектури. – Київ, 2025.

Робота присвячена системі батьківського контролю для веб-сервісів у вигляді розширення для браузера. Інструментом реалізації розроблюваної системи є бібліотеки та фреймворки мови програмування JavaScript.

Ключові слова: Хром розширення, безпека, API, JavaScript.

SUMMARY

Khomenko V. R. Development of a system of parental control of web services.

Master's Thesis in Specialty 125 "Cybersecurity and Information Security", Specialization "Security of Information and Communication Systems". - Kyiv National University of Construction and Architecture, Kyiv, 2025.

The work is devoted to the system of parental control of web services in the form of a browser extension. The tools for implementing the developed system are libraries and frameworks of the JavaScript programming language.

Keywords: Chrome extension, security, API, JavaScript.

РЕЗЮМЕ (SUMMARY) *до кваліфікаційної випускової роботи здобувача*	ПІБ *здобувача українською та англійською мовами* Хоменко Влад Романович Khomenko Vlad Romanovich		
ЗВО	Київський національний університет будівництва і архітектури		
Тема *(українською та англійською)*	Система батьківського контролю для веб-сервісів Parental Control System for Web Services		
Освітній ступінь	Магістр		
Факультет	Автоматизації і інформаційних технологій		
Випускова кафедра	Кібербезпеки та комп'ютерної інженерії		
Спеціальність	125 «Кібербезпека та захист інформації»		
Освітня програма	Безпека інформаційних і комунікаційних систем		
Керівник	Гуменний Дмитро Олександрович		
Обсяг роботи:	*Пояснювальна записка, стор.*	*Розділів*	*Презентація, кількість слайдів*
	98	5	18
Розділ 1	Аналіз предметної області та постановка задачі		
Розділ 2	Архітектура системи		
Розділ 3	Проєктування бази даних		
Розділ 4	Практична реалізація та тестовий приклад		
Розділ 5	Висновки		
Висновки по роботі	Система відповідає заявленій меті і може використовуватися в реальних умовах		
Ключові слова: Keywords:	Хром розширення, безпека, API, JavaScript		

Здобувач _____/ _____

Керівник _____/ _____

ЗМІСТ

Вступ

У сучасному цифровому віці, де доступ до Інтернету став необхідністю, важливо забезпечити безпеку та контроль за веб-діяльністю дітей. Розвиток інтернет-технологій та поширення веб-сервісів створює потребу в надійних і ефективних інструментах батьківського контролю.

Ця робота присвячена розробці системи батьківського контролю для веб-сервісів, яка дозволить батькам забезпечувати безпеку та контроль за онлайн-активністю своїх дітей. Система буде включати в себе хром-розширення, сервер, авторизацію, базу даних, клієнтську частину та саме браузерне розширення, яке буде взаємодіяти з веб-сервісами.

Розробка системи буде відбуватись за допомогою мови програмування JavaScript (TypeScript) у середовищі Visual Studio Code.

Перелік умовних позначень

API – application programming interface

ІС – інформаційна система

БД – база даних

IDE – інтегроване середовище розробки (Integrated Development Environment)

SPA – Single-Page Application — односторінковий веб-додаток

SSR – Server-Side Rendering — серверний рендеринг

UI – User Interface — інтерфейс користувача

UX – User Experience — користувацький досвід

UML – Unified Modeling Language — уніфікована мова моделювання

ERD – Entity-Relationship Diagram — діаграма зв’язків сутностей

CRUD – Create, Read, Update, Delete — базові операції з даними

JWT – JSON Web Token — токен автентифікації у веб-системах

ORM – Object-Relational Mapping — об’єктно-реляційне відображення

SQL – Structured Query Language — мова структурованих запитів

DB – Database — база даних

СУБД / DBMS – Система управління базами даних / Database Management System

JSON – JavaScript Object Notation — формат обміну даними

HTTP – HyperText Transfer Protocol — протокол передачі гіпертекстуIDE – Integrated Development Environment — інтегроване середовище розробки

RTS – Real-Time Synchronization — синхронізація в реальному часі

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної області

Дана дипломна робота спрямована на розробку системи батьківського контролю для веб-сервісів, що має на меті забезпечити батькам можливість ефективно контролювати та обмежувати доступ своїх дітей до інтернет-ресурсів.

Предметною областю даного проекту є сфера онлайн-безпеки та контролю доступу до веб-ресурсів. З поширенням веб-сервісів та збільшенням їхнього впливу на поведінку та розвиток дітей, виникає необхідність в інструментах для забезпечення безпеки та контролю..

Стрімкий розвиток інформаційного суспільства та масова цифровізація всіх сфер життя зумовили суттєве зростання ролі веб-сервісів у процесах соціалізації та формування світогляду дітей і підлітків. Інтернет перестав бути факультативним інструментом — він став ключовим середовищем для навчання, комунікації, розваг та саморозвитку. Разом із тим збільшилися й ризики, пов’язані з доступом неповнолітніх до небажаного, шкідливого чи деструктивного контенту. Саме тому предметною областю даного дослідження є сфера забезпечення онлайн-безпеки **дітей**, зокрема технології контролю та регулювання доступу до веб-ресурсів.

У межах цього дослідження розглядається створення системи батьківського контролю для веб-сервісів, яка функціонуватиме у вигляді браузерного розширення та дозволятиме здійснювати аутентифікацію користувачів, створювати та редагувати персоналізовані списки дозволених (whitelist) і заборонених (blacklist) веб-ресурсів, а також контролювати інтернет-активність дітей. Такий підхід відповідає сучасним тенденціям розвитку цифрових сервісів та забезпечує зручний механізм інтеграції контролю безпосередньо у повсякденну роботу браузера.

Необхідність у подібних системах обумовлена тим, що традиційні методи виховання та нагляду вже не можуть гарантувати безпеку дитини в онлайн-середовищі, яке постійно змінюється та характеризується відкритістю і доступністю величезної кількості інформації. Дослідження міжнародних організацій, таких як EU Kids Online та UNICEF, демонструють стабільну

тенденцію до зростання кількості інцидентів, пов’язаних із кібербулінгом, онлайн-шахрайством. Це створює необхідність у розробці інструментів, здатних автоматизувати процес фільтрації веб-контенту, а також забезпечити прозорість і контрольованість інтернет-активності.

Основними аспектами предметної області є:

– визначення та регулювання рівня доступу дітей до інформаційних ресурсів;

– моніторинг часу, проведеного в мережі;

– фільтрація контенту за категоріями ризику;

– забезпечення безпеки персональних даних користувачів;

– адаптивність системи до швидко змінюваного ландшафту веб-технологій.

Сучасні системи батьківського контролю повинні враховувати не лише необхідність обмеження доступу, але й вимоги до юзабіліті, масштабованості, а також відповідність нормативно-правовим актам щодо обробки даних неповнолітніх (зокрема GDPR, COPPA тощо). Важливо підкреслити, що інструменти фільтрації повинні бути гнучкими, конфігуруватися відповідно до вікових особливостей та рівня цифрової компетентності дитини, а також забезпечувати можливість швидкого оновлення бази заборонених ресурсів у відповідь на еволюцію загроз. [25][9]

До переваг веб-орієнтованих систем контролю можна віднести [10]:

– можливість централізованого управління доступом;

– автоматичне застосування фільтрації незалежно від сайту, на який переходить користувач;

– потенціал інтеграції з хмарними сервісами для синхронізації налаштувань між пристроями;

– гнучкість у створенні профілів користувачів із різними рівнями обмежень.

Недоліки предметної області пов’язані, перш за все, з необхідністю постійного оновлення механізмів фільтрації через швидкі зміни веб-контенту, а також з можливістю обходу обмежень технічно підкованими користувачами. Окремим викликом є баланс між контролем і конфіденційністю, оскільки

надмірний моніторинг може порушувати етичні норми та права дитини на приватність.

Таким чином, система батьківського контролю для веб-сервісів є актуальним і затребуваним інструментом, що відповідає потребам сучасного цифрового суспільства. Її розробка спрямована на створення ефективного, адаптивного та безпечного механізму регулювання доступу дітей до інтернет-ресурсів з метою мінімізації ризиків та формування безпечного інформаційного середовища.

1.2 Аналіз об’єкта дослідження

Об’єктом дослідження у цій роботі є система батьківського контролю для веб-сервісів, що функціонує як інтерактивний інструмент забезпечення безпеки дітей у цифровому середовищі. Така система належить до класу складних інформаційних систем, у яких поєднуються механізми моніторингу, фільтрації, авторизації, обробки подій та адміністрування користувацьких правил. Її основною метою є реалізація багатоаспектного контролю за веб-активністю неповнолітніх і створення безпечного інформаційного середовища для їх розвитку.

Аналіз об’єкта дослідження спирається на сучасні тенденції у сфері кібербезпеки, сімейної цифрової політики та інтернет-менеджменту. У міру того, як інтернет-платформи та веб-сервіси вдосконалюються і адаптуються під потреби користувачів, збільшується й кількість ризиків, з якими зустрічаються діти. Зокрема, це небажаний контент, шахрайські ресурси, кібербулінг, соціальна інженерія, надмірна екранна активність та інші загрози. Тому системи батьківського контролю виконують важливу роль як інструменти превенції та управління цифровою поведінкою.

Ключові аспекти об’єкта дослідження

- Забезпечення безпеки в Інтернеті

Одним із головних викликів сучасного цифрового середовища є наявність великої кількості небезпечного та віковоневідповідного контенту. До нього належать [9]:

- матеріали з насильством або агресією,
- шахрайські сайти, спрямовані на отримання персональних даних,

- ресурси з пропагандистським або деструктивним змістом.

Система батьківського контролю має забезпечувати своєчасне **в**иявлення та блокування подібних матеріалів, використовуючи методи фільтрації, категоризації та семантичного аналізу веб-ресурсів. Це дозволяє мінімізувати ризик випадкового або цілеспрямованого доступу до шкідливої інформації.

- Механізми обмеження доступу

Система повинна дозволяти батькам формувати індивідуальні політики доступу, включно з:

- блокуванням окремих сайтів;
- забороною переходу на певні категорії веб-ресурсів (розваги, соцмережі, азартні ігри тощо);
- обмеженням часу користування інтернетом;
- встановленням правил поведінки в мережі для різних дітей або профілів.

Гнучкість системи обмежень є критично важливою, оскільки цифрові звички та вікові особливості дітей можуть суттєво відрізнятися.

- Надання доступу до відібраних дозволених ресурсів

Одним із найефективніших підходів у сфері веб-безпеки є режим білих списків (whitelist-mode). У цьому режимі:

- за замовчуванням блокується весь інтернет,
- дозволяється лише перелік чітко визначених сайтів, попередньо схвалених батьками.

Такий підхід є найбільш суворим, але водночас і найбезпечнішим, що робить його актуальним для дітей молодшого віку або ситуацій, де необхідний максимальний рівень контролю. [25]

Аналіз об’єкта дослідження також передбачає розгляд базових технічних механізмів, що використовуються сучасними системами батьківського контролю. Таблиця 1.1 узагальнює ключові методи та описує принципи їх функціонування.

Таблиця 1.1. Методи контролю та принципи їх роботи

Метод	Опис	Принцип роботи
Моніторинг	Відслідковування відвіданих сайтів	Система записує всі відвідані веб-сайти та активність користувачів, згідно з налаштуваннями батьків
Фільтрація контенту	Фільтрація доступу до небажаного контенту на основі заданих параметрів	Система блокує доступ до визначеного контенту на основі попередньо встановлених правил та налаштувань
Фільтрація за принципом виключень	Батьки визначають конкретні веб-сайти, які є безпечними та дозволеними для використання дитиною	Система блокує всі веб-сайти за замовчуванням, але ті, що знаходяться в списку білих сайтів, будуть доступні

Об’єкт дослідження - система батьківського контролю - є багатокомпонентним технічним рішенням, яке поєднує аналіз поведінки користувача, контроль доступу, фільтрацію даних та адміністрування політик безпеки. Її актуальність визначається зростанням ролі цифрових технологій у житті дітей та необхідністю ефективних механізмів захисту в умовах постійного розширення веб-простору.

1.3 Опис предмету дослідження

Предметом даного дослідження є браузерне розширення для Google Chrome, призначене для реалізації механізмів батьківського контролю у веб-середовищі. Створення розширення як інструменту контролю зумовлене необхідністю забезпечення безпечного доступу дітей до мережі Інтернет, а також зростаючою актуальністю застосування мікросервісних, хмарних та браузерних технологій у сфері інформаційної безпеки. Даний підхід узгоджується з рекомендаціями Mozilla

та Google щодо використання WebExtensions API як універсальної платформи безпеки браузерів [1, 2].

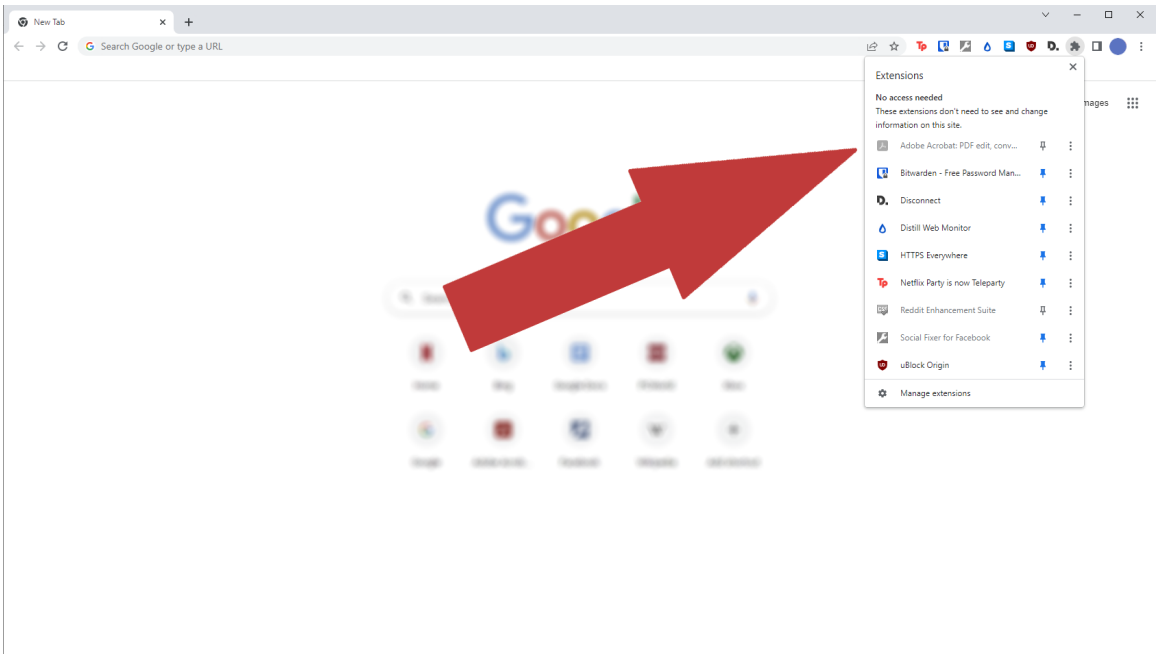

Рисунок 1.1. Хром розширення

Google Chrome — це багатоплатформний веб-браузер, розроблений компанією Google у 2008 році. Завдяки високій швидкості роботи, стабільності та потужній екосистемі інструментів для розробників Chrome став домінуючим браузером на світовому ринку. Станом на 2024–2025 роки його ринкова частка становить понад 65%, що робить його найбільш популярним браузером у світі. (Google Developers. Chrome Identity API)

Ключові особливості Chrome:

- Підтримка сучасних веб-стандартів (HTML5, ES2023, CSS4, WebAssembly).
- Потужний механізм безпеки, включно із sandboxing, ізоляцією процесів та системою виявлення шкідливих сайтів Safe Browsing.
- Велика екосистема розширень, що дозволяє доповнювати браузер новими можливостями без необхідності встановлення окремих програм.
- **К**росплатформність: Chrome доступний на Windows, macOS, Linux, Android, iOS та ChromeOS.

Google Chrome є найбільш поширеним браузером у світі, що підтверджується аналітичними звітами Google Developers [13]. Вбудовані механізми sandboxing та Safe Browsing забезпечують додатковий рівень захисту [13, 16].

З огляду на це, вибір Chrome як платформи для реалізації системи батьківського контролю є доцільним і технологічно обґрунтованим.

Розширення Chrome як інструмент програмної реалізації

Розширення браузера (browser extension) — це невеликий програмний модуль, що інтегрується у браузер з метою розширення або модифікації його функціональності [2]. Їхня архітектура базується на використанні:

- JavaScript/TypeScript
- HTML та CSS
- Chrome Extensions API
- Manifest V3 (сучасний стандарт розробки розширень)

Функціональні можливості розширень дозволяють:

- перехоплювати та аналізувати мережеві запити;
- змінювати контент веб-сторінок;
- взаємодіяти з системами авторизації та хмарними сервісами;
- зберігати налаштування та дані користувача;
- відслідковувати поведінку браузера та блокувати небажані ресурси.

Саме Chrome Extensions API забезпечує розробникам доступ до сотень функцій для створення контролюючих, безпекових та фільтраційних механізмів.

Причини вибору розширення Chrome для реалізації системи

Вибір браузерного розширення як основного програмного рішення пояснюється низкою факторів [2]:

1. Широке охоплення аудиторії. Chrome використовується більшістю користувачів, що робить розширення доступним для великої кількості сімей.
2. Безкоштовність розробки та використання. Google надає інструменти розробки безкоштовно, а публікація в Chrome Web Store не потребує плати за підтримку.

3. Підтримка на різних пристроях. Chrome синхронізує розширення між ПК, телефонами та планшетами, дозволяючи застосовувати батьківські обмеження у будь-якому середовищі.
4. Безпека. Розширення працюють в ізольованому середовищі та відповідають політикам Google щодо захисту приватності.

Таким чином, браузерне розширення Chrome є оптимальним технічним середовищем для реалізації системи батьківського контролю.

API (Application Programming Interface) — це набір інструментів і правил, що визначають спосіб взаємодії між програмними компонентами. У межах цієї роботи API виконує такі ключові функції:

- забезпечення комунікації між розширенням і серверною частиною системи;
- передавання користувацьких даних, зокрема списків сайтів, налаштувань облікового запису тощо;
- авторизація користувачів через JWT або OAuth; [17]
- отримання політик доступу в реальному часі;
- синхронізація налаштувань між пристроями.

Використання API гарантує масштабованість системи: у майбутньому можливо додати мобільний застосунок, веб-панель адміністратора чи окремі модулі машинного навчання.

База даних як ядро системи зберігання інформації

База даних (БД) — це організована структура, що дозволяє надійно зберігати, оновлювати й обробляти великі обсяги взаємопов’язаної інформації. У системі батьківського контролю БД використовується для:

- зберігання облікових записів користувачів;
- зберігання списків дозволених (whitelist) та заборонених сайтів (blacklist);
- зберігання журналів активності;
- логування подій;
- зберігання налаштувань браузерного розширення;
- синхронізації даних між пристроями й сеансами.

У межах цієї роботи використовується PostgreSQL — реляційна система керування базами даних (РСУБД), що є однією з найбільш стабільних і потужних у світі. [7]

Переваги вибору PostgreSQL:

- підтримка складних типів даних;
- висока надійність та транзакційність;
- широкі можливості масштабування;
- відкрита ліцензія (повністю безкоштовне використання);
- сумісність з сучасними backend-фреймворками (Nest.js, Django, Spring).

Таким чином, база даних виступає ключовим компонентом системи, забезпечуючи стійкість, структурованість та безпечність збереження користувацької інформації.

Предмет дослідження - браузерне розширення Chrome у поєднанні з серверною частиною та базою даних - являє собою гнучку, масштабовану й технологічно сучасну систему. Її архітектура дозволяє забезпечити високий рівень безпеки та ефективний батьківський контроль, використовуючи переваги Chrome Extensions API, хмарних сервісів та реляційної бази даних PostgreSQL.

Система є прикладом інтегрованого підходу до вирішення проблеми безпечного доступу дітей до інтернет-ресурсів, поєднуючи інструменти з боку браузера, серверної логіки та інфраструктури зберігання даних.

1.4 Аналіз актуальності

У сучасному цифровому середовищі інтернет став невід’ємною частиною повсякденного життя як дорослих, так і дітей. Зростання кількості онлайн ресурсів, активне використання мобільних пристроїв та доступність інформації з будь-якого місця створюють як нові можливості для розвитку, так і суттєві ризики. Саме тому питання контролю онлайн активності дітей перетворюється на одну з ключових проблем для сучасних сімей. [16]

Останні роки характеризуються стрімким збільшенням часу, який діти та підлітки проводять у мережі. Вони користуються соціальними мережами, переглядають відео, грають в онлайн ігри, використовують навчальні та

розважальні ресурси. Разом із цим значно підвищується ймовірність контакту з небажаним, шкідливим або психологічно небезпечним контентом. У мережі поширені матеріали, що можуть містити насильство, пропаганду, фейки, азартні ігри, а також ресурси, що сприяють залежності або впливають на формування небажаних поведінкових моделей.

Актуальність систем батьківського контролю посилюється й тим, що діти часто мають недостатньо сформовані навички критичного мислення та саморегуляції. Вони не завжди здатні самостійно визначити межі безпечної онлайн поведінки або вчасно розпізнати ризики. Це покладає додаткову відповідальність на батьків, які прагнуть забезпечити дитині збалансоване та безпечне цифрове середовище.

До ключових факторів, що підкреслюють важливість розробки та впровадження систем батьківського контролю, можна віднести такі:
1.) стрімке зростання кількості веб-сервісів, мобільних застосунків та онлайн платформ, що можуть містити небажаний або непристойний контент
2.) підвищення інтересу дітей до соціальних мереж, де існує ризик кібербулінгу, небажаних контактів і витоку персональних даних
3.) усвідомлення батьками впливу інтернету на психологічний стан та розвиток дитини, що спонукає їх контролювати інформаційне середовище
4.) розвиток технологій, які дозволяють створювати ефективні інструменти моніторингу, фільтрації контенту, обмеження екранного часу та аналітики активності
5.) зростання кількості досліджень, що підтверджують негативні наслідки безконтрольного використання інтернету дітьми, включаючи зниження концентрації уваги, порушення сну, підвищення тривожності та формування залежної поведінки

З огляду на ці фактори, системи батьківського контролю вже не сприймаються як допоміжний інструмент, а стають необхідною складовою цифрової безпеки сім’ї. Вони дозволяють батькам адаптувати доступ до контенту відповідно до віку та психологічного стану дитини, встановлювати здорові

обмеження часу користування пристроями та своєчасно реагувати на потенційні ризики.

Проблематика захисту дітей у цифровому середовищі є предметом активних наукових досліджень у галузях інформаційної безпеки, соціології та педагогіки. Значна кількість публікацій присвячена аналізу ризиків, що виникають у процесі використання Інтернету дітьми, а також розробці підходів до обмеження доступу до небажаного контенту. Розгляд наукових праць дозволяє виявити як ефективні методи батьківського контролю, так і обмеження існуючих підходів.

У статті Livingstone і Helsper «Balancing opportunities and risks in teenagers' use of the Internet» автори досліджують негативні наслідки безконтрольного доступу дітей до вебресурсів та підкреслюють зростання кіберризиків, пов'язаних із насильницьким контентом, порнографією та онлайн-шахрайством [18]. Перевагою дослідження є ґрунтовний статистичний аналіз та охоплення великої вибірки респондентів. Водночас робота має соціологічний характер і не містить практичних рекомендацій щодо побудови програмних систем захисту.

У роботі Dinh та співавторів «A content filtering approach based on machine learning for parental control systems» запропоновано використання алгоритмів машинного навчання для класифікації вебконтенту [19]. Перевагою підходу є висока точність фільтрації за умови достатнього навчального набору даних. Недоліком є значні апаратні вимоги та складність впровадження в браузерних системах, що працюють у реальному часі.

У дослідженні Xu, Zhang і Li «Design of an intelligent parental control system based on web content analysis» розглядається використання семантичного аналізу вебсторінок для виявлення небажаного змісту [20]. Основною перевагою роботи є детальна структуризація системи за модулями, що спрощує подальший розвиток архітектури. Недоліком є потреба у великих обчислювальних ресурсах для аналізу текстових даних.

У науковій статті Ko та ін. «Online risks and parental mediation» автори аналізують вплив методів батьківського контролю на психологічний стан дітей [21]. Зроблено висновок, що надмірний контроль може негативно позначитися на

довірі між дітьми та батьками. Однак робота не пропонує технічних рішень щодо реалізації систем захисту.

У дослідженні Zhang і Gupta «Web usage monitoring for child safety» основну увагу приділено моніторингу вебактивності користувачів як методу виявлення потенційно небезпечної поведінки [22]. Перевагою підходу є можливість раннього виявлення загроз, а недоліком — ризик порушення конфіденційності персональних даних.

У публікації Rahman та ін. «URL-based filtering techniques for parental control systems» розглянуто методи фільтрації на основі URL-адрес [23]. Перевагою цього підходу є мінімальне навантаження на систему, що особливо важливо для браузерних рішень. Водночас метод не завжди ефективний у випадках використання коротких посилань та динамічних редиректів.

У праці Park і Kim «Access control models for child protection in web applications» запропоновано концепцію контролю доступу з використанням ролей користувачів [24]. Основною перевагою є чітке розмежування прав доступу, однак реалізація таких моделей у браузерних розширеннях ускладнена через обмеження API.

У звіті UNICEF «Children in a Digital World» наведено глобальний аналіз загроз для дітей в Інтернеті [25]. Документ підтверджує необхідність створення інструментів контролю та захисту в цифровому середовищі, однак не пропонує технічних реалізацій.

Аналіз наукових робіт дозволяє виділити такі ключові положення:

1. Проблема безпеки дітей в Інтернеті є актуальною та підтвердженою науково.
2. Більшість досліджень є або теоретичними, або такими, що важко адаптувати до браузерних систем.
3. Жодне дослідження не пропонує універсального рішення, яке поєднує простоту використання, прозорість блокування та синхронізацію налаштувань.

Отже, аналіз актуальності показує, що потреба у створенні надійних і зручних у використанні систем батьківського контролю є вкрай високою. Такі системи

сприяють формуванню безпечного інформаційного простору, підтримують здоровий розвиток дітей, допомагають запобігти негативному впливу небажаного контенту та забезпечують відповідальне використання цифрових технологій у сучасному світі.

1.5 Стан вже існуючих рішень

На сучасному ринку цифрових технологій представлено велику кількість систем, інструментів та браузерних розширень, орієнтованих на забезпечення батьківського контролю. Їх основна мета полягає у створенні безпечного середовища для дітей під час використання інтернету, а також у наданні батькам можливості відстежувати активність дитини, контролювати її доступ до веб-ресурсів та застосунків і своєчасно реагувати на потенційні ризики. З розвитком інтернет-сервісів такі рішення стають необхідністю для багатьох сімей, оскільки дозволяють компенсувати нестачу контролю, що виникає через масове використання дітьми смартфонів, планшетів та персональних комп’ютерів.

Сучасні рішення представлені у вигляді окремих програм, вбудованих інструментів у операційні системи, а також браузерних розширень. Браузерні розширення особливо популярні завдяки простоті встановлення, невеликому розміру, швидкому оновленню та можливості працювати без складних налаштувань. Вони інтегруються в інтерфейс браузера і забезпечують контроль у реальному часі. Найпоширенішими функціями таких рішень є:

1. блокування доступу до небажаних або шкідливих веб-сайтів, включаючи ресурси із забороненим, відвертим чи агресивним контентом
2. встановлення обмежень часу використання інтернету або окремих сайтів, що дозволяє формувати здорові цифрові звички
3. створення білого списку сайтів, доступ до яких дозволено завжди
4. фільтрація за ключовими словами для автоматичного виявлення небажаних матеріалів
5. моніторинг активності дитини, включаючи історію переглядів, тривалість перебування на сайтах, взаємодію з окремими ресурсами

6. створення графіку обмежень, який дозволяє батькам гнучко керувати доступом залежно від часу доби, навчального розкладу чи вихідних днів

Попри широкий функціонал, вже існуючі рішення мають низку обмежень. Частина з них працює за передплатою, інші не підтримують повний контроль над усіма сайтами або не мають функції відстеження активності. Деякі розширення можуть бути незручними у використанні, містити складний інтерфейс або потребувати додаткових налаштувань, що ускладнює їхнє впровадження для пересічних користувачів.

З розвитком веб-технологій виникає потреба у створенні нових рішень, що базуються на сучасних підходах до обробки даних, фільтрації контенту, адаптивного аналізу поведінки користувача та забезпечення стабільності роботи. Покращення можливостей мов програмування, поява нових фреймворків та API відкривають шлях до створення більш точних і гнучких систем батьківського контролю, які здатні працювати швидше й ефективніше, ніж попередні покоління програм.

Під час розробки браузерного розширення для батьківського контролю слід враховувати такі ключові аспекти:

1. якість розширення

Розширення повинно мати інтуїтивно зрозумілий інтерфейс, прості налаштування, відсутність зайвих елементів і логічну структуру. Важливо забезпечити зручну навігацію для користувачів, які не мають високого технічного досвіду.

2. функціональність

Рішення повинно містити всі необхідні можливості для контролю контенту, фільтрації сайтів, створення розкладу та відстеження активності. Необхідно забезпечити баланс між широким функціоналом і простотою користування.

3. продуктивність і стабільність

Розширення не повинно уповільнювати роботу браузера, викликати затримки завантаження сторінок або конфліктувати з іншими встановленими плагінами.

4. безпека даних

Оскільки система працює з персональною інформацією дитини, важливо забезпечити захист даних, відсутність передачі конфіденційної інформації третім сторонам і мінімізацію збору даних.

5. можливість регулярного оновлення

Шкідливий контент постійно змінюється, тому розширення повинно мати механізми оновлення фільтрів, списків сайтів та логіки перевірки.

Для аналізу існуючих інструментів розглянемо два популярні рішення: BlockSite та Web Blocker. Ці розширення добре відомі серед користувачів та широко використовуються для базових функцій фільтрації веб-контенту.

Таблиця 1.2. Порівняння існуючих рішень

Рішення	BlockSite	Web Blocker
Блокування сайтів	Так	Так
Білий список сайтів	Так	Ні
Фільтрація за ключовими словами	Так	Ні
Моніторинг використання	Ні	Ні
Розклад блокування	Так	Так
Вартість	Платний (безкоштовно під час пробного періоду)	Безкоштовний

Як показує аналіз, наявні рішення виконують базові функції блокування контенту та створення розкладу, проте вони мають обмеження. BlockSite пропонує розширені можливості, але є платним продуктом, що може бути недоступним для частини користувачів. Web Blocker, своєю чергою, безкоштовний, однак має

значно меншу кількість функцій, відсутність білого списку та відсутність фільтрації за ключовими словами, що робить його менш ефективним.

Таким чином, попри широкий вибір інструментів для батьківського контролю, ринок залишається відкритим для нових рішень, орієнтованих на сучасні технології, розширений функціонал, зручність використання та адаптивність до конкретних потреб сімей. Розробка нового розширення дозволить заповнити цю нішу, запропонувати сучасний підхід до контролю активності та забезпечити вищий рівень безпеки для дітей у цифровому середовищі.

1.6 Визначення цілей дослідження та постановка задачі

Метою даного дослідження є розробка та практична реалізація програмного забезпечення для батьківського контролю у середовищі веб-браузера. Така система повинна забезпечувати можливість ефективного моніторингу та керування відвідуванням веб-сайтів дітьми, а також створювати безпечне цифрове середовище шляхом блокування небажаного контенту, фільтрації веб-ресурсів та зберігання інформації про активність користувачів.

Актуальність поставленої мети обумовлена швидким розвитком інформаційних технологій, збільшенням кількості веб-ресурсів та розширенням інтернет-доступу серед дітей та підлітків. У цих умовах традиційні способи контролю втрачають свою ефективність, а батькам стає все важче забезпечувати належний рівень безпеки. Тому створення інструменту, який поєднує сучасні веб-технології, гнучкі налаштування та зручність використання, є важливим і своєчасним завданням.

Для досягнення поставленої мети у роботі визначено комплекс основних завдань, що охоплює всі етапи розробки програмного забезпечення:

- створення браузерного розширення, яке інтегруватиметься у роботу веб-браузера та виконуватиме функції контролю доступу
- розробка клієнтської частини програми, що забезпечить взаємодію користувача з системою, дозволить налаштовувати списки сайтів, переглядати дані, змінювати параметри доступу

- розробка серверної частини програмного забезпечення, яка відповідатиме за обробку інформації, взаємодію між компонентами системи та зберігання даних
- створення бази даних для фіксації інформації про користувачів, їхні налаштування, списки дозволених чи заблокованих сайтів та інші параметри
- підключення сервера до бази даних і забезпечення стабільного обміну даними, що дасть змогу системі працювати надійно та без затримок

Кожне із зазначених завдань є важливим етапом, який формує загальну архітектуру програмного комплексу. Їх вирішення дозволяє створити цілісну систему, здатну працювати в реальних умовах, забезпечувати контроль активності та зберігати інформацію для подальшого аналізу.

Об’єктом дослідження у даній роботі є процес контролю веб-сервісів батьками, що включає обмеження доступу до певних ресурсів, відстеження дій у мережі та управління цифровими звичками дітей. Об’єкт відображає широку сферу взаємодії користувачів із веб-контентом, а також механізми впливу на цю взаємодію за допомогою програмних засобів.

Предметом дослідження є розробка та впровадження програмного забезпечення для батьківського контролю в браузерах, що реалізується за допомогою мови програмування JavaScript та пов'язаних з нею фреймворків і бібліотек. Саме вибір цих технологій дозволяє забезпечити кросплатформеність, інтеграцію з браузером та можливість швидкої обробки даних. Предмет охоплює як технічні аспекти створення програмного продукту, так і організацію його роботи, логіку взаємодії компонентів, принципи фільтрації веб-контенту та методи підвищення безпеки.

Таким чином, сформована мета та поставлені завдання є основою для подальшої розробки проєкту. Вони визначають напрям дослідження та створюють чіткий план реалізації функціональної системи батьківського контролю, здатної вирішувати актуальні проблеми цифрової безпеки дітей.

2. АРХІТЕКТУРА СИСТЕМИ

Архітектура програмної системи визначає не лише її внутрішню структуру, але й можливості подальшого розвитку. Грамотно спроєктована архітектура забезпечує гнучкість, масштабованість та підтримуваність програмного продукту протягом усього життєвого циклу.

Масштабованість є однією з ключових вимог до сучасних програмних рішень. Вона означає здатність системи працювати ефективно при збільшенні кількості користувачів, обсягу даних або числа запитів. Архітектура визначає, наскільки просто можна додати нові модулі, розширити функціональність або оптимізувати підвищене навантаження.

У випадку системи батьківського контролю масштабування має особливу актуальність з огляду на можливість одночасного використання на декількох пристроях, синхронізації правил блокування в реальному часі та постійні запити до сервера щодо перевірки дозволених ресурсів.

Архітектурні рішення, які були застосовані у системі, зменшують зв’язаність компонентів і підвищують незалежність модулів. Наприклад, модуль керування обліковими записами не залежить безпосередньо від модуля обробки BlockList, що дозволяє змінювати або надбудовувати компоненти незалежно один від одного. Використання шаблону MVC дозволяє логічно розмежувати візуальну частину системи, бізнеслогіку та обробку запитів, що забезпечує прозору структуру коду.

Архітектура системи безпосередньо впливає на рівень надійності програмного забезпечення. Програмні рішення, які не мають чіткої архітектурної моделі, зазвичай важко масштабуються, складні у підтримці та швидко втрачають стабільність у процесі розвитку.

У контексті системи батьківського контролю особливо важливо забезпечити безперебійну роботу механізмів авторизації, обробки правил блокування та синхронізації даних. Навіть часткова відмова сервісу може призвести до некоректної роботи фільтрації та неналежного контролю доступу. Запроваджена архітектура дозволяє локалізувати помилки в межах окремих компонентів, не

порушуючи роботу всієї системи. Наприклад, у випадку відмови сервісу ведення журналів робота механізму блокування продовжується без змін.

Архітектурна модель також полегшує реалізацію відмовостійкості шляхом застосування резервних механізмів, зокрема кешування даних та автоматичного відновлення сесій користувачів.

Архітектура програмного забезпечення є ключовим елементом у процесі створення ефективної, масштабованої та надійної системи батьківського контролю. У розробці даного проєкту використано архітектурний шаблон MVC, який упродовж багатьох років зарекомендував себе як один із найбільш логічних та структурованих підходів до побудови вебсистем. Використання цього шаблону дозволяє розділити логіку програми на окремі сутності, що значно полегшує розробку, тестування, подальшу підтримку та розширення функціональності системи.

Шаблон MVC складається з трьох основних компонентів

- модель
- уявлення
- контролер

Кожен із цих компонентів має власну зону відповідальності, завдяки чому досягається чіткий розподіл обов'язків між елементами системи.

Модель є центральною частиною системи, оскільки містить структуру даних та логіку їх обробки. У межах даного проєкту модель реалізована у вигляді класів BlockList, Auth та Account. Кожен з цих класів відповідає за окремий фрагмент системної логіки. Клас Auth відповідає за обробку операцій, пов'язаних з автентифікацією користувача, зокрема створення нового облікового запису, вхід у систему та завершення сеансу. Клас Account містить бізнеслогіку, пов'язану з інформацією про користувача, включно з можливістю редагування профілю, збереження персональних налаштувань і взаємодією з іншими компонентами програми. Клас BlockList відповідає за роботу зі списком сайтів, що належать до категорії заборонених або дозволених, а також за обробку запитів, пов'язаних зі зміною цього списку.

Система спроектована та розроблена згідно шаблону проектування MVC. (Рисунок 4.1).

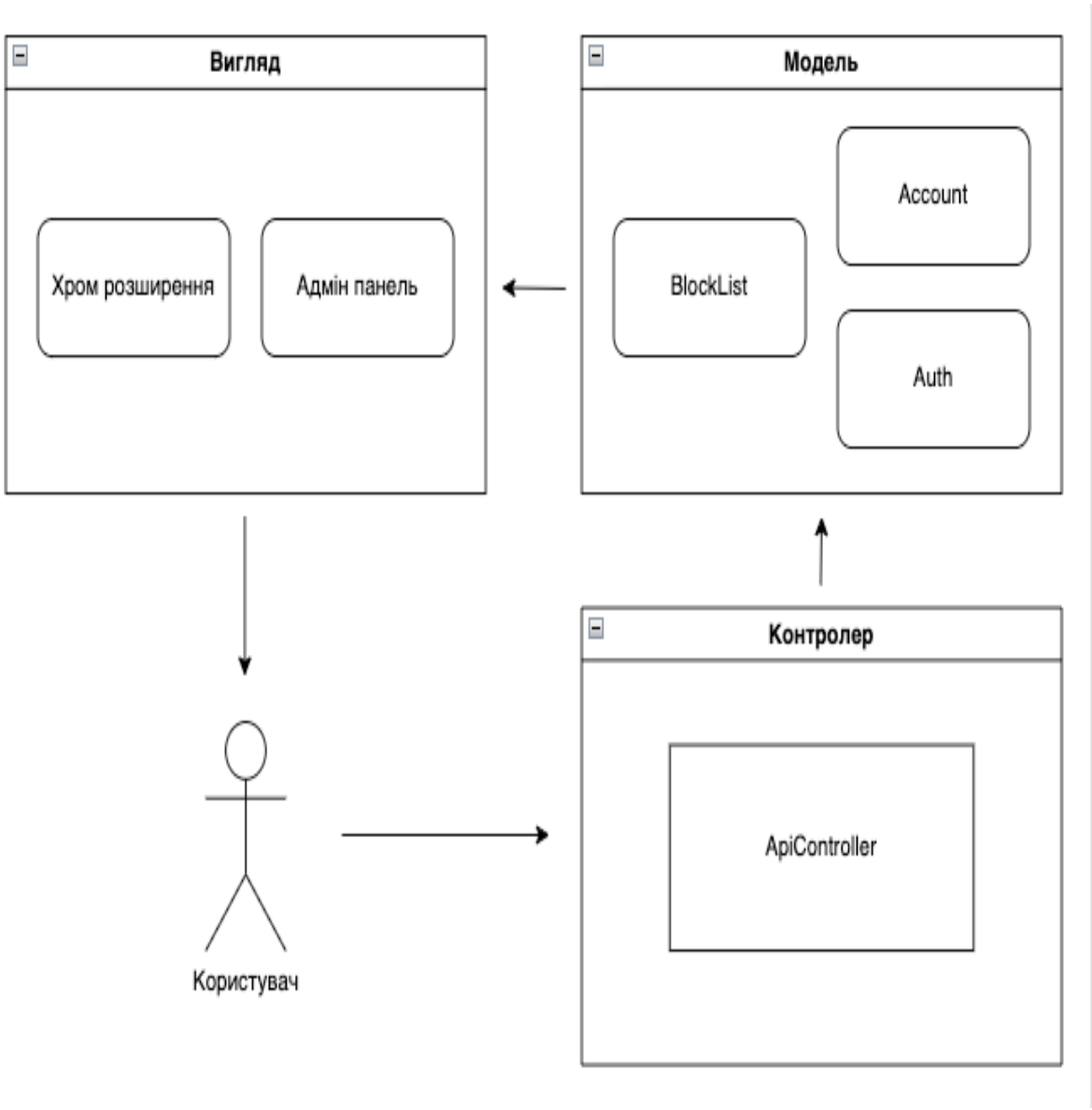

Рисунок 4.1 Шаблон проектування MVC

Контролер ApiController виконує роль посередника між моделлю та клієнтом. Він приймає HTTPзапити, обробляє їх відповідно до визначеної логіки та повертає відповідь. Контролер не містить бізнеслогіки, а лише координує роботу моделі, передає дані та забезпечує коректний формат відповіді. Завдяки цьому забезпечується незалежність моделі від способів представлення даних та взаємодії з користувачем.

Усі можливі HTTP-запити, що обробляються контролером, наведено у таблиці 4.1. Кожен маршрут відповідає певній функціональності в межах системи.

Запити, пов'язані з автентифікацією, дозволяють користувачам створювати облікові записи, входити до системи та виходити з неї. Запити для роботи з акаунтом надають можливість переглядати або редагувати інформацію користувача. Запити, що пов'язані зі списком блокувань, відображають можливість користувача додавати нові елементи до списку заблокованих ресурсів або навпаки видаляти їх.

Таблиця 4.1 – Обробка HTTP-запитів у контролері системи

Шлях	HTTP-метод доступу	Аргументи методу контролера
/auth/sign-up	POST	SignUpBodyDto
/auth/sign-in	POST	SignInBodyDto
/auth/sign-out	POST	-
/account	GET,	-
/account	PATCH	PatchAccountDto
/block-list	GET	-
/block-list/:item	POST	BlockItemDto
/block-list/:item	DELETE	id

Використання архітектури MVC забезпечує гнучкість у процесі розробки та підтримки системи. Завдяки чіткому розподілу на моделі, контролери та представлення кожен модуль можна легко модернізувати без впливу на інші частини програми. Це особливо важливо для систем батьківського контролю, які потребують стабільної роботи, високого рівня безпеки та можливості швидкого розширення функціоналу, наприклад додавання нових механізмів фільтрації або інтеграції з іншими сервісами.

У перспективі така архітектура дозволяє масштабувати проєкт, змінювати структуру моделі, додаючи нові сутності, доповнювати контролери новими маршрутами та розширювати клієнтську частину без необхідності суттєвого переписування основної логіки. Саме завдяки цим перевагам шаблон MVC був обраний як оптимальна основа для побудови системи батьківського контролю для веб-сервісів.

2.1 Концептуальна схема системи

Концептуальна схема системи є узагальненим представленням логічної структури програмного комплексу та визначає головні елементи, їх призначення і взаємодію між собою. Така схема не заглиблюється у технічні деталі реалізації, проте дозволяє сформувати цілісне уявлення про роботу системи, взаємозв'язки між її підсистемами та основні інформаційні потоки. У контексті розробки системи батьківського контролю концептуальна схема відіграє важливу роль, оскільки дозволяє заздалегідь визначити логіку взаємодії клієнтської частини, серверної інфраструктури та бази даних, забезпечуючи узгодженість і структурованість на всіх етапах проєктування.

Основними елементами концептуальної схеми є користувач, браузерне розширення, сервер додатку та база даних. Кожен з цих компонентів виконує окрему функцію, але повноцінна робота системи можлива лише за умов їх тісної взаємодії. Браузерне розширення виступає клієнтською частиною, що безпосередньо взаємодіє з користувачем та інтернетсайтами, тоді як серверна частина відповідає за обробку логіки, перевірку прав доступу, збереження стану системи та забезпечення безпеки даних.

Першим елементом концептуальної схеми є інтерфейс користувача, який представлений у вигляді браузерного розширення Chrome. Це розширення містить усі інструменти, необхідні для взаємодії батьків із системою: форми реєстрації та авторизації, модулі управління списками дозволених і заблокованих сайтів, а також засоби перегляду стану облікового запису. Інтерфейс повинен бути максимально простим та інтуїтивним, оскільки саме він визначає ефективність використання системи у реальних умовах. Розширення працює безпосередньо у браузері та має можливість перехоплювати URLзапити, тому воно виконує важливу роль первинного фільтра, що аналізує кожну відвідувану вебсторінку.

Наступним елементом є сервер додатку. Саме він виступає центральним компонентом, що координує роботу системи. Сервер приймає запити від браузерного розширення, здійснює аутентифікацію та авторизацію користувачів, обробляє логіку роботи списків блокування, а також виконує валідацію даних. На

серверній частині реалізована логіка інтерпретації команд із браузера, перевірка, чи належить конкретний сайт до списку дозволених або заблокованих, і переведення цього рішення назад у клієнтську частину. Сервер забезпечує надійність, узгодженість та цілісність даних, навіть якщо користувач працює з кількох пристроїв.

База даних є ключовим компонентом концептуальної системи. У рамках системи використовується PostgreSQL як сучасне, надійне та продуктивне рішення для зберігання структурованої інформації. У базі даних зберігаються облікові записи користувачів, налаштування їхніх профілів, інформація щодо списків заблокованих сайтів, записи дозволених ресурсів, а також службові дані, необхідні для внутрішньої роботи системи. Застосування бази даних дозволяє зберігати стійкий стан системи, забезпечує можливість швидкого доступу до інформації та гарантує її актуальність. Крім того, використання серверної БД дозволяє підтримувати кілька пристроїв одного користувача, що є важливим у сучасних умовах багатоплатформності.

Взаємодія між компонентами системи здійснюється шляхом обміну інформацією через HTTPзапити. Браузерне розширення ініціює запит до сервера у випадках, коли користувач здійснює вхід у систему, змінює параметри свого профілю, додає сайт до списку блокування або навпаки вилучає його. Сервер приймає ці дані, проводить обробку, звертається до бази даних у разі необхідності та повертає відповідь у стандартизованому форматі JSON. Завдяки цьому забезпечується уніфікована структура обміну інформацією, яка дозволяє масштабувати систему та полегшує подальший розвиток функціональності.

Важливим елементом концептуальної схеми є інформаційні потоки. В межах системи існує два основних потоки даних. Перший ініціюється користувачем і передає інформацію від браузерного розширення до сервера, наприклад дані про створення нового елемента у списку блокування. Другий потік відбувається у зворотному напрямку і містить від сервера до розширення інформацію про успішність виконання операції, поточний стан списку сайтів, статус автентифікації

та інші параметри. Завдяки чітко визначеним каналам обміну даними система забезпечує надійний контроль і високу точність виконання поставлених задач.

Загальна архітектура системи, демонструє логічну узгодженість взаємодії компонентів і показує, яким чином окремі частини об’єднані в єдине функціональне середовище. Концептуальна схема дозволяє сформувати загальне уявлення про роботу системи, визначити її межі, оцінити взаємозалежності і потенційні вузькі місця, а також є базою для створення детальніших схем, таких як фізична модель даних або діаграми взаємодії.

Таким чином, концептуальна схема системи виступає важливим етапом проєктування, що формує фундамент для подальшої розробки. Вона визначає компоненти, їх роль та способи взаємодії, забезпечує логічність структури та слугує відправною точкою для побудови більш деталізованих моделей, необхідних для реалізації програмного забезпечення батьківського контролю.

На Рисунку 4.3 можна ознайомитися з концептуальною схемою системи:

Рисунок 4.3 Концептуальна схема системи

2.2 Функціональна схема системи

Функціональна схема системи (Functional Flow Block Diagram — FFBD) є важливою частиною опису архітектури, оскільки вона демонструє не лише загальну логіку роботи програмного забезпечення, але й деталізує порядок виконання операцій та інформаційні потоки між окремими компонентами. На відміну від концептуальної схеми, яка відображає структуру, FFBD фокусується саме на динаміці, тобто на тому, як система працює під час взаємодії з користувачем.

Функціональна схема показує послідовність дій від моменту запуску браузера та активації розширення до отримання відповіді від сервера і застосування політик блокування. Вона будується на принципах функціональної декомпозиції, де кожен процес поділяється на логічні етапи, що дозволяють чітко визначити відповідальність кожного модуля.

Основні принципи побудови функціональної схеми

1. Послідовність виконання операцій.
   FFBD чітко визначає порядок, у якому запускаються функції: автентифікація, завантаження налаштувань, перевірка списку блокування, обробка запитів до сервера та ін.
2. Логічна залежність між блоками.
   Кожен наступний крок виконується лише після завершення попереднього. Це дозволяє моделювати реальну поведінку системи у браузері.
3. Розмежування функціональності між клієнтською та серверною частинами.
   Функціональна схема демонструє, яка логіка обробляється на стороні розширення, а яка — сервером чи базою даних.
4. Підкреслення ролі користувача.
   Оскільки це система батьківського контролю, саме користувач (батько/опікун) ініціює більшу частину процесів: авторизацію, редагування списку блокування, перегляд активності тощо.

Опис основних функціональних блоків системи

Нижче подається розширений опис ключових етапів, що відображені на функціональній схемі (Рисунок 4.4):

1. Ініціалізація браузерного розширення

Після запуску браузера система автоматично завантажує фоновий скрипт розширення. Він виконує початкову перевірку стану користувача: чи є активна сесія, чи потрібно повторно пройти авторизацію, чи необхідно оновити локальні дані зі списку блокування.

2. Перевірка авторизації

Розширення звертається до сервера через API-запит. Можливі результати:

- користувач авторизований — завантажуються налаштування;
- користувач не авторизований — відкривається сторінка входу.

3. Отримання даних користувача

Після успішної авторизації сервер повертає:

- інформацію про обліковий запис;
- актуальний список заблокованих сайтів;
- налаштування облікового запису.

Ці дані зберігаються у локальному сховищі браузера для швидкого доступу.

4. Перехоплення веб-запитів

Ключовий компонент розширення — система перехоплення URL-запитів. Вона працює на рівні браузера та аналізує:

- домен,
- піддомен,
- конкретний шлях сторінки,
- правила блокування.

5. Зіставлення URL з блок-листом

На цьому етапі розширення перевіряє, чи входить URL у список заборонених. Якщо URL збігається з одним із елементів — спрацьовує логіка блокування.

6. Застосування політики блокування

Можливі дії:

- повне блокування сайту з показом сторінки-попередження;
- перенаправлення на інший ресурс;
- логування спроби відкриття сайту для подальшого аналізу.

7. Синхронізація з сервером

Розширення періодично оновлює:

- список блокування;
- дані користувача;

- зміни, внесені через вебінтерфейс або мобільний пристрій.

Це забезпечує постійну актуальність інформації.

8. Управління обліковим записом

Через клієнтську частину користувач може:

- редагувати профіль;
- додавати або видаляти сайти з блок-листа;
- вмикати/вимикати контроль;
- переглядати історію блокувань.

Функціональна схема системи демонструє цілісний життєвий цикл взаємодії з програмним забезпеченням: від моменту входу користувача в систему до безпосередньої фільтрації веб-контенту.

На Рисунку 4.4 представлено структуровану послідовність усіх цих етапів, що дозволяє візуально зрозуміти логіку їх взаємозв’язків та забезпечує можливість подальшої оптимізації або модернізації системи.

Окремим результатом є усвідомлення того, що функціональна схема не лише служить інструментом документування, але й виконує роль методологічної основи для подальших етапів проєктування. Таким чином, функціональна схема системи дозволяє впевнено стверджувати, що архітектура рішення є добре спроєктованою, придатною до масштабування та адаптації до нових вимог. Вона створює основу для безпечної, стабільної та ефективної роботи системи батьківського контролю в реальних умовах експлуатації.

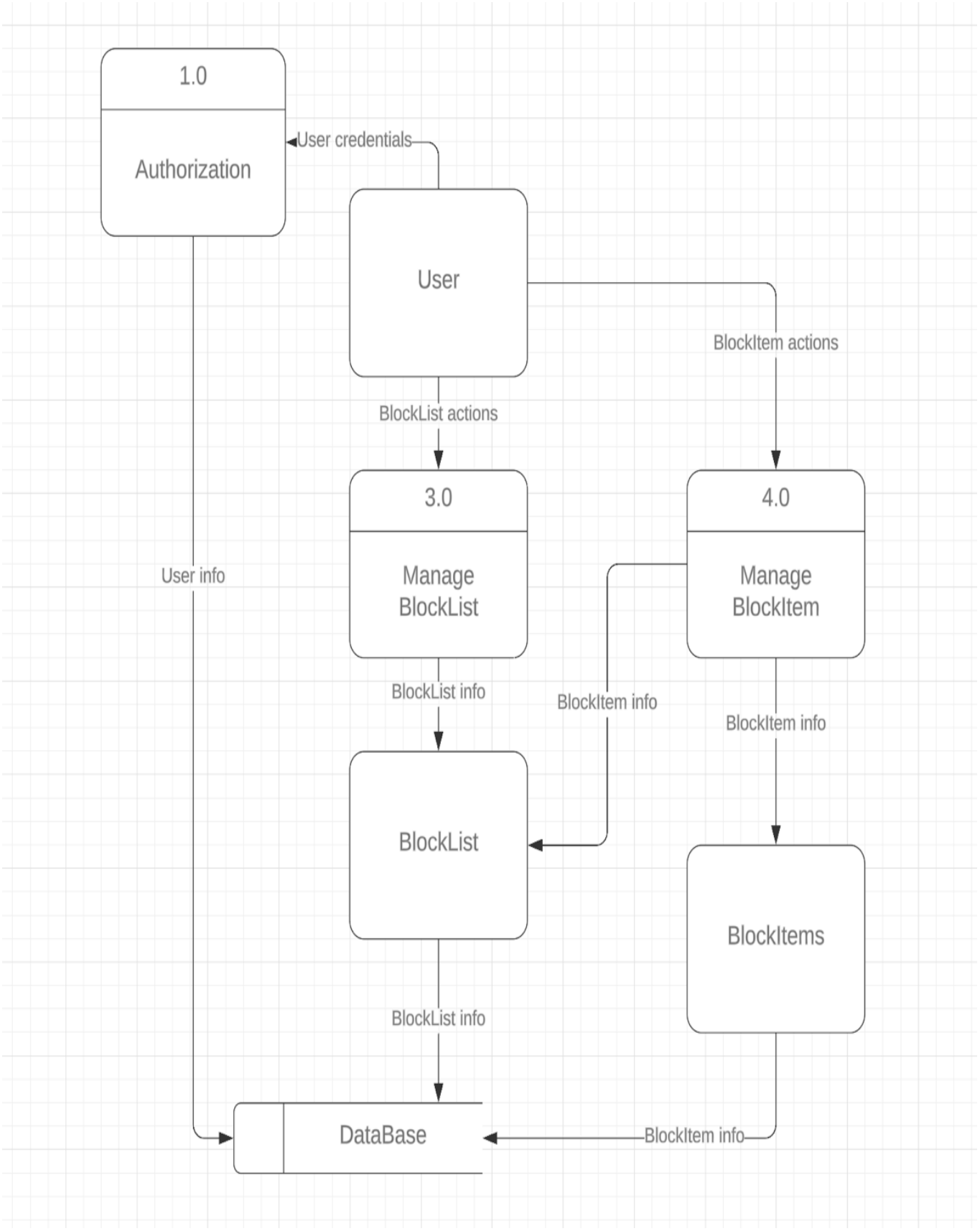

Рисунок 4.4 Функціональна схема системи

2.3 Принципи модульності та слабкого зв’язування

Архітектура модульної системи ґрунтується на принципі слабкого зв’язування компонентів, що передбачає мінімальну кількість залежностей між функціональними одиницями програмного забезпечення. Такий підхід дозволяє змінювати або замінювати окремі частини системи без необхідності повного перепроєктування всієї структури.

У системі батьківського контролю модульність застосовується для відокремлення таких підсистем:

- обробки запитів користувача;
- керування автентифікацією;
- роботи зі списками обмежень;
- взаємодії з базою даних;
- інтерфейсу користувача.

Кожен модуль реалізує власний набір функцій і не містить логіки інших підсистем. Це забезпечує високу підтримуваність проєкту, оскільки розробник може працювати з одним модулем, не порушуючи цілісність системи.

Особливу важливість має інверсія залежностей, коли реалізація бізнеслогіки не залежить від конкретних технічних засобів доступу до даних. Такий підхід спрощує зміну ORM-рішень, структури БД або серверного фреймворка без значних змін у коді.

Принцип слабкого зв’язування також підвищує тестованість. Кожен компонент може бути протестований автономно, що спрощує виявлення помилок та підвищує надійність програмного забезпечення.

2.4 Масштабування та розвиток архітектури

Масштабування програмної системи передбачає здатність адаптувати архітектуру до збільшення навантаження або розширення функціональності. Система батьківського контролю спроєктована таким чином, щоб розширення не вимагало повного переписування коду.

Вертикальне масштабування дозволяє підвищити продуктивність шляхом використання потужніших серверів, тоді як горизонтальне масштабування

реалізується розгортанням кількох вузлів обробки запитів. Передбачена можливість балансування навантаження між серверами та використання окремих служб для обробки фонових задач.

Під час проєктування архітектури враховано можливість:

- зовнішньої авторизації через сторонні сервіси;
- підключення мобільних клієнтів;
- реалізації систем аналітики;
- підтримки кількох профілів у межах одного облікового запису.

2.5 Архітектурні стилі побудови веб-систем

Архітектура програмної системи визначає принциповий підхід до реалізації логіки, обробки даних та взаємодії між компонентами. У сучасних веб-системах застосовуються різні архітектурні стилі, серед яких найпоширенішими є монолітна архітектура, мікросервісна архітектура, клієнт-серверна модель, подієво-орієнтована архітектура та сервісно-орієнтована архітектура.

Монолітна архітектура передбачає, що всі функціональні частини системи реалізуються в одному програмному продукті. Такий підхід є відносно простим у реалізації, однак він має суттєві недоліки з точки зору масштабованості, підтримки та розширюваності. Будь-яка зміна в системі вимагає повторної збірки та розгортання всього додатку.

Мікросервісна архітектура передбачає поділ системи на незалежні сервіси, які взаємодіють між собою через визначені інтерфейси. Кожен сервіс може мати власну базу даних, середовище виконання та життєвий цикл. Перевагою такого підходу є гнучкість масштабування, тоді як недоліком – зростання складності адміністрування.

У рамках даної роботи використовується клієнт-серверна архітектура з чітким розподілом ролей: клієнтська частина відповідає за взаємодію з користувачем, серверна частина виконує обробку бізнес-логіки та взаємодію з базою даних.

Подібний підхід є оптимальним для систем батьківського контролю, де відбувається регулярна синхронізація налаштувань між сервером та клієнтом. Всі критичні рішення приймаються на сервері, що підвищує захищеність системи та запобігає зловживанням.

Використання архітектури MVC дозволило ізолювати презентаційний рівень від бізнес-логіки та моделі даних. Це дозволяє забезпечити гнучкість змін, легкість тестування та можливість незалежної розробки окремих компонентів.

2.6 Взаємодія компонентів системи в реальному часі

Однією із ключових вимог до системи є оперативна реакція на зміну налаштувань блокування. Зміни, які виконуються на сервері, повинні відображатися у браузерному розширенні без потреби перезавантаження браузера або повторної авторизації.

Для досягнення цього використовується механізм синхронізації через REST API. При кожному запуску розширення відбувається первинне завантаження конфігурації користувача. У випадку зміни налаштувань система оновлює локальний стан клієнта.

Взаємодія між клієнтом та сервером враховує можливість виникнення похибок зв'язку, тимчасових затримок та відсутності інтернет-з’єднання. У таких випадках система зберігає попередній стан фільтрації до моменту відновлення зв'язку, що гарантує стабільність блокування.

2.7 Аналітичне обґрунтування вибору методів та технологій

Використання браузерного розширення як інструмента реалізації системи батьківського контролю обумовлене його прямою інтеграцією у середовище перегляду веб-контенту. Такий підхід дозволяє здійснювати фільтрацію трафіку без потреби встановлення окремого програмного забезпечення або втручання в операційну систему. Відповідно до рекомендацій Mozilla та Google, розширення мають повний доступ до API навігації, запитів, локального сховища та управління вкладками, що необхідно для реалізації політик доступу [1][13].

Крім того, архітектура WebExtensions забезпечує єдиний підхід до розширень у різних браузерах, що робить розроблену систему масштабованою та придатною до використання не лише в Google Chrome [1].

Метод фільтрації за принципом блокування доменів та ключових слів є базовим інструментом більшості сучасних систем батьківського контролю. Згідно з аналітичними матеріалами OWASP, фільтрація контенту є ефективним підходом до зменшення загроз пов'язаних із фішингом, небезпечними ресурсами та шкідливими вебсайтами [9].

Метод «білого списку» є найбільш суворим, однак забезпечує максимальний рівень контролю, оскільки всі ресурси блокуються за замовчуванням, а доступ дозволяється лише до дозволених сайтів. Такий підхід рекомендований для молодших вікових груп (Google Safe Browsing, 2024).

Застосування фільтрації за ключовими словами дозволяє реагувати на динамічний контент, що не має стабільних доменних адрес, що є типовим для соціальних мереж та відеохостингів (Cloudflare, 2024).

Архітектура «клієнт–сервер» була обрана через необхідність централізованого збереження налаштувань користувача та синхронізації між різними пристроями. Такий підхід широко застосовується в сучасних вебсистемах та рекомендований як найбільш стабільний та масштабований [5].

Використання REST API обумовлене його простотою, доступністю та підтримкою більшістю клієнтських платформ. REST є стандартом де-факто для веб-застосунків [10].

PostgreSQL обрано як СУБД через підтримку транзакційності, масштабовності та високого рівня безпеки. PostgreSQL реалізує механізм MVCC, що гарантує цілісність даних у паралельних операціях [7].

NestJS забезпечує модульну структуру та централізовану обробку запитів, що критично важливо для масштабування [6].

Prisma дозволяє використовувати типізовану модель доступу до даних, що істотно знижує ризик SQL-помилок [8].

3. ПРОЕКТУВАННЯ БАЗИ ДАНИХ

3.1 Опис вибору СУБД

База даних - набір впорядкованої та структурованої інформації, що зберігається в електронному вигляді. Завдяки БД (Базам даних) набагато зручніше виконувати управління, коригування, оновлення, контроль та впорядкування.

Однією з ключових переваг баз даних є оперативність їхньої роботи при додаванні та використанні інформації. Завдяки вдосконаленим алгоритмам, які використовуються у їхній структурі, знаходження потрібних даних займає лише кілька секунд. Крім того, бази даних забезпечують внутрішню взаємодію даних: зміна у одній частині може вплинути на інші — це сприяє ефективнішій та швидшій обробці інформації.

Бази даних для веб-сайтів функціонують як спеціально організовані зв'язані між собою структури. У них зберігається весь необхідний контент для оптимальної роботи сайту: дані користувачів, цінові пропозиції, асортимент товарів.

Створення запитів до баз даних часто виконується за допомогою мови структурованих запитів (Structured Query Language, SQL). SQL надає можливість додавати, змінювати та видаляти інформацію, що зберігається у таблицях.

Системи управління базами даних (СУБД) представляють собою комплекс програмних та мовних інструментів, спрямованих на створення, зберігання, керування та використання баз даних. Вони забезпечують ефективне та надійне зберігання даних і надають користувачам зручний інтерфейс для роботи з ними.

Роль СУБД в управлінні даними полягає в структурованому доступі до інформації, у мінімізації дублювання даних та забезпеченні цілісності та безпеки даних.

Використання СУБД дозволяє впровадити структурований підхід до управління інформацією, що забезпечує зменшення дублювання даних, зниження ймовірності помилок та підвищення загальної ефективності обробки інформації. На відміну від звичайних файлів, бази даних забезпечують логічний зв’язок між таблицями, уніфікований механізм витягування та оновлення інформації, а також

підтримку механізмів транзакційності, що гарантують узгодженість системи навіть у випадку збоїв.

Швидкість обробки інформації є однією з ключових властивостей сучасних БД. Навіть при значних обсягах даних система здатна виконувати складні запити за лічені секунди, що є критично важливим для інформаційних сервісів, орієнтованих на реальний час роботи. Оптимізація виконання SQL-запитів, використання індексів і кешування дозволяють значно підвищити продуктивність системи, особливо при одночасному доступі великої кількості користувачів.

Для веборієнтованих програмних продуктів база даних виконує роль ядра збереження контенту. Вона містить інформацію про облікові записи користувачів, параметри доступу, дозволені ресурси, налаштування безпеки та конфігурації системи. У разі змін у обліковому записі або списку сайтів відповідні коригування миттєво фіксуються у БД, після чого стають доступними для використання всіма компонентами системи.

Управління базами даних здійснюється через системи управління базами даних, які являють собою комплекс програмних та інтерфейсних засобів для створення, адміністрування та використання баз даних. СУБД забезпечують користувачів інструментами виконання запитів, керування доступом, резервного копіювання та відновлення. Однією з ключових функцій СУБД є забезпечення цілісності інформації, коли будь-яка операція над даними перевіряється на правильність і узгодженість.

Історія розвитку СУБД демонструє поступову еволюцію від простих файлових структур до складних об’єктно-орієнтованих і розподілених систем. Перші СУБД, що з’явилися у 1960-х роках, базувалися на ієрархічній та мережевій моделях, що обмежувало гнучкість структури даних. Реляційні бази даних, що з’явилися в 1970-х роках, стали переломним етапом у розвитку технологій зберігання даних, оскільки дозволили формалізувати залежності між таблицями та стандартизувати мову роботи з даними через SQL.

Подальший розвиток об’єктно-орієнтованих СУБД був спрямований на подолання обмежень реляційної моделі, пов’язаних зі складними структурами

даних. Об’єктно-реляційні СУБД, у свою чергу, об’єднали переваги обох підходів, запропонувавши розширення реляційних таблиць типами даних, функціями та механізмами обробки складних структур.

На сучасному етапі активно розвиваються хмарні СУБД, які дозволяють отримувати доступ до бази даних через Інтернет без необхідності підтримки серверної інфраструктури на власному обладнанні. Такий підхід значно спрощує масштабування та технічне обслуговування.

Перші системи управління базами даних з'явилися у 1960-х роках та були спрямовані на зберігання даних у ієрархічній або мережевій структурах. У 1970-х роках з'явилися реляційні системи управління базами даних, які залишаються найпоширенішими до сьогодні. У 1980-х роках стали розвиватися об'єктно-орієнтовані системи управління базами даних, що підходять для зберігання даних про складні об'єкти. Важливим етапом у історії СУБД була розробка мови структурованих запитів SQL у 1970-х роках. Останні роки свідчать про розвиток хмарних систем управління базами даних, які надають користувачам доступ до даних через Інтернет. Управління даними систем управління базами даних здійснюється шляхом ефективного та систематизованого зберігання інформації. Наприклад, у онлайн-магазині СУБД ефективно управляють інформацією про товари, замовлення та клієнтів, забезпечуючи швидкий доступ до потрібної інформації. Інструменти вилучення та оновлення дозволяють керувати даними в базі, забезпечуючи ефективну взаємодію з інформацією. Оптимізація запитів гарантує швидкодію системи, особливо у сферах, де необхідний моментальний доступ, наприклад, у медичних організаціях для швидкого отримання медичних записів.

Системи управління базами даних класифікуються за різними критеріями, включаючи модель даних (реляційні, ієрархічні, мережеві), функціональне призначення (оперативні та аналітичні), розподіл (централізовані та розподілені), тип використання (SQL-орієнтовані та NoSQL) і ліцензію (відкриті та пропрієтарні). Знання переваг і недоліків кожного типу допомагає вибрати найкращий варіант залежно від конкретних потреб проєкту.

Окрему увагу необхідно приділити питанням безпеки, оскільки система батьківського контролю працює з персональними даними. Сучасні СУБД підтримують контроль доступу, шифрування, резервне копіювання, аудит операцій та захист від несанкціонованого доступу. Дані повинні бути доступні лише авторизованим користувачам і зберігатися у захищеному вигляді.

В таблиці 4.5 наведено таблицю з порівнянням типів СУБД:

Таблиця 4.5 порівняння типів СУБД

Тип СУБД	Опис	Переваги	Недоліки
Ієрархічні	Організовують дані у вигляді ієрархії, де кожен запис має батька та нащадків	– Простота в моделюванні зв’язків.	– Обмеження в гнучкості структури даних.
Мережеві	Дозволяють встановлювати складні зв’язки між даними, створюючи структуру, схожу на граф	– Підтримка складних і взаємопов’язаних даних.	– Складність в управлінні структурою і підтримці запитів. – Складнощі в підтримці та обслуговуванні.
Реляційні	Засновані на теорії відносин і таблиць. Кожна таблиця представляє окреме відношення.	– Простота у використанні та моделюванні даних.	– Продуктивність може страждати за великих обсягів даних. – Складність у поданні складних ієрархій даних.

Об’єктно-орієнтовані	Працюють з об’єктами, де дані представлено у вигляді об’єктів із методами та властивостями.	– Зручність у моделюванні складних об’єктних структур. – Підтримка успадкування, поліморфізму та інкапсуляції.	– Складність у міграції з традиційних СУБД. – Обмежена підтримка мови SQL.
Об’єктно-реляційні	Комбінують риси реляційних і об’єктно-орієнтованих СУБД.	– Поєднання переваг реляційних і об’єктно-орієнтованих моделей. – Підтримка більш складних структур даних.	– Додаткова складність у проєктуванні.

Системи управління базами даних (СУБД) пропонують різноманітні засоби безпеки, серед яких контроль доступу, шифрування даних, забезпечення цілісності даних та резервне копіювання. Контроль доступу обмежує можливість доступу до даних лише авторизованим користувачам, шифрування захищає дані від несанкціонованого використання, а підтримка цілісності та резервне копіювання гарантують надійність і збереження даних.

PostgreSQL є потужною об'єктно-реляційною системою управління базами даних з відкритим вихідним кодом, що поєднує в собі переваги реляційних баз даних та об'єктно-орієнтованого програмування. Вона використовує клієнт-серверну архітектуру з окремими процесами для кожного підключення, а також застосовує мультиверсійний контроль паралельності (MVCC) для забезпечення узгодженості даних і Write-Ahead Logging (WAL) для забезпечення стійкості до збоїв і можливості відновлення.

В таблиці 4.6 наведено порівняння PostgreSQL з іншими СУБД:

Таблиця 4.6 порівняння PostgreSQL з іншими СУБД

Критерій	PostgreSQL	MySQL	SQLite
Вартість	Безкоштовно	Безкоштовно	Безкоштовно
Відкритість	Відкритий вихідний код	Відкритий вихідний код	Відкритий вихідний код
Підтримувані мови програмування	SQL, PL/pgSQL, Python, Perl, Java, C, C++, Ruby	SQL, MySQL, PHP, Python, Perl	SQL, Python, C++, Java, PHP, Ruby
Підтримувані операційні системи	Windows, Linux, macOS, FreeBSD	Windows, Linux, macOS	Windows, Linux, macOS, Solaris, HP-UX, AIX, z/OS
Підтримувані архітектури	x86, x86-64, ARM, RISC-V	x86, x86-64, ARM	x86, x86-64, ARM, RISC-V
Продуктивність	Хороша	Хороша	Хороша
Надійність	Висока	Висока	Висока
Безпека	Висока (підтримка SSL, контроль доступу, шифрування)	Середня (підтримка SSL, контроль доступу)	Низька (обмежені можливості безпеки)

Функціональність	Підтримка складних запитів, індексів, тригерів	Підтримка складних запитів, індексів, тригерів, переглядів	Обмежена підтримка індексів, тригерів, переглядів

Завдяки тому, що PostgreSQL належить до об'єктно-реляційних СУБД і має відкритий вихідний код, ця система надає великі можливості для налаштування та адаптації. Підтримка широкого набору функцій забезпечує гнучкість у реалізації різних сценаріїв використання. Висока продуктивність, надійність та рівень безпеки гарантують ефективну і захищену роботу з даними, що робить PostgreSQL оптимальним вибором для системи батьківського контролю для веб-сервісів.

3.2 Опис всіх таблиць та полів бази даних

Діаграми «сутність-зв’язок» (ERD) призначені для розробки моделей даних та забезпечують стандартний засіб визначення даних і відношень між ними. Фактично за допомогою ERD здійснюється деталізація сховищ даних проектованої системи, а також документуються сутності системи та засоби їхньої взаємодії, включаючи ідентифікацію об'єктів, важливих для предметної області (сутності), властивостей цих об'єктів (атрибутів) і їхніх відношень з іншими об'єктами (зв’язків). ERD використовується як засіб моделювання даних, що забезпечує графічне відображення структури бази даних на концептуальному рівні. Вона дозволяє виявити основні сутності, їх атрибути, визначити типи зв’язків та обмеження цілісності. Основне призначення ERD полягає у створенні зрозумілої моделі бази даних, що полегшує перехід від етапу проєктування до безпосередньої реалізації в СУБД.

Сутність являє собою множину екземплярів реальних або абстрактних об'єктів (людей, подій, станів, ідей, предметів і т. ін.), що мають спільні атрибути або характеристики. Будь-який об'єкт системи може бути представлений лише однією сутністю, що повинна бути унікально ідентифікована. При цьому ім'я сутності повинно відображати тип або клас об'єкту, а не його конкретний

екземпляр (наприклад, КНИГА, а не назва конкретної книги). Сутність у контексті ERD являє собою абстрактний об’єкт предметної області, який має власний набір користувацьких або службових атрибутів та може формувати зв’язки з іншими сутностями. Прикладом сутності може бути користувач системи, список заблокованих сайтів або обліковий запис. Кожна сутність в межах системи повинна бути унікально ідентифікована, для чого використовується первинний ключ.

Відношення відображають взаємозв’язки між сутностями. Вони можуть бути типу один до одного, один до багатьох або багато до багатьох. У базах даних ці зв’язки реалізуються шляхом використання зовнішніх ключів. Важливою властивістю ERD є можливість однозначно встановити залежності між таблицями ще до створення фізичної моделі бази даних.

Відношення в самому загальному вигляді являє собою зв'язок між двома і більшою кількістю сутностей. Найменування відношення здійснюється за допомогою граматичного звороту дієслова (має, визначає, може володіти і т. ін.)

Кожна сутність володіє однім або декількома атрибутами, що однозначно ідентифікують кожен примірник (екземпляр) сутності. При цьому будь-який атрибут може бути визначений як ключовий.

Головною перевагою ERD в контексті реляційних баз даних є їхня тісна відповідність зі структурою самих баз даних. Реляційні бази даних побудовані на таблицях, у яких зберігаються структуровані дані – окрема таблиця для кожного об'єкта, а взаємозв'язки між об'єктами реалізуються за допомогою первинних та зовнішніх ключів. ERD забезпечують простий та інтуїтивно зрозумілий спосіб візуалізації цих ключових елементів та їхніх взаємозв'язків, що сприяє безперешкодному переходу від проектування до впровадження та обслуговування бази даних.

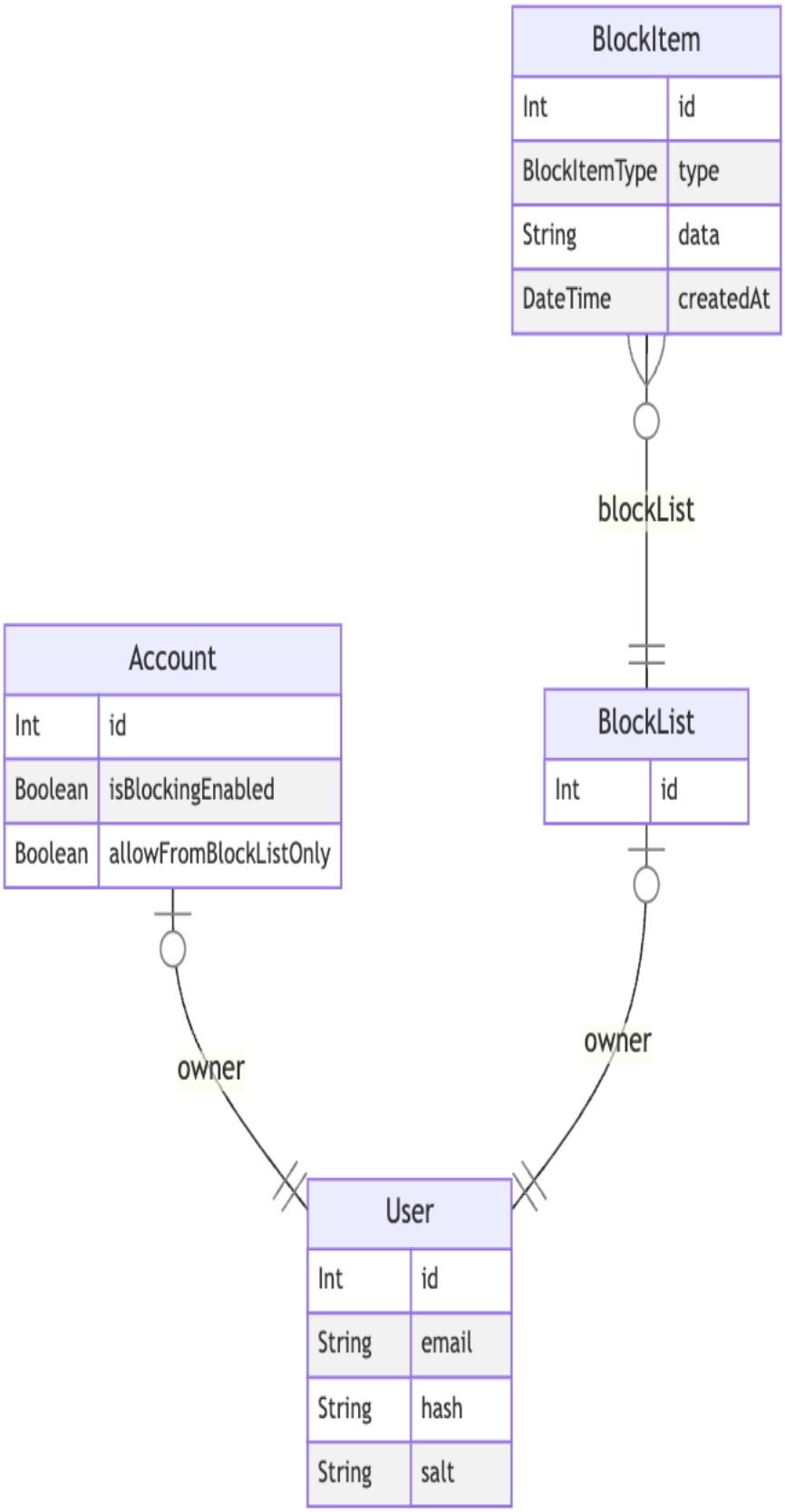

Рисунок 4.6 ERD бази даних

Таблиця 4.7 Опис таблиць та полів БД

Таблиця	Назва поля	Тип	Опис поля
User	id	Int	PK, ідентифікатор
	email	String	Унікальна пошта користувача
	hash	String	Хеш пароля користувача
	salt	String	Сіль для хешування пароля
	account	Account?	Зв’язок з таблицею Account
	blockList	BlockList?	Зв’язок з таблицею BlockList
Account	id	Int	PK, ідентифікатор
	ownerId	Int	Унікальний ідентифікатор власника
	owner	User	Зв’язок з таблицею User
	isBlockingEnabled	Boolean	Флаг активації блокування
	allowFromBlockListOnly	Boolean	Флаг режиму “білого списку”
BlockList	id	Int	PK, ідентифікатор
	ownerId	Int	Унікальний ідентифікатор власника
	owner	User	Зв’язок з таблицею User

	items	BlockItem[]	Масив елементів списку
BlockItem	id	Int	PK, ідентифікатор
	blockListId	Int	Ідентифікатор списку
	blockList	BlockList	Зв’язок з таблицею BlockList
	type	BlockItemType	Тип елемента списку
	data	String	Дані елемента списку

Таблиця User є основною сутністю системи та відповідає за збереження даних користувачів. Вона є центральною ланкою всієї БД, оскільки через неї пов’язані інші таблиці.

Поля таблиці User:

id – ціле числове поле, первинний ключ таблиці. Використовується для унікальної ідентифікації кожного користувача в системі.

email – текстове поле, що зберігає адресу електронної пошти користувача. Це поле має бути унікальним для кожного запису і використовується як логін для входу в систему.

hash – текстове поле, що містить хеш пароля. Пароль у відкритому вигляді не зберігається з міркувань безпеки.

salt – випадковий рядок, що використовується для генерації хешу пароля. Цей механізм захищає від атак типу rainbow table.

account – поле зв’язку із таблицею Account, що визначає асоціацію один до одного.

blockList – поле зв’язку з таблицею BlockList, що визначає асоціацію користувача зі списком обмежень.

Таблиця User реалізує принцип мінімального зберігання персональних даних та враховує вимоги до інформаційної безпеки, оскільки не зберігає відкриті паролі.

Таблиця Account є логічним розширенням сутності User та містить інформацію, що безпосередньо визначає режим роботи системи.

Поля таблиці Account:

id – первинний ключ таблиці.

ownerId – зовнішній ключ, що вказує на користувача, якому належить акаунт.

owner – асоціація з таблицею User, що формує зв’язок один до одного.

isBlockingEnabled – логічний прапорець, що визначає чи активований режим блокування.

allowFromBlockListOnly – логічний параметр, що відповідає за режим білого списку.

Таблиця Account використовується для відокремлення облікового запису від інформації про самого користувача. Це забезпечує гнучкість при розвитку системи, наприклад у майбутньому можливо реалізувати кілька профілів в межах одного облікового запису.

Таблиця BlockList відповідає за збереження списку обмежувальних правил користувача.

Поля таблиці BlockList:

id – первинний ключ.

ownerId – зовнішній ключ на таблицю User.

owner – асоціація з користувачем.

items – поле типу масиву, яке представляє множину елементів списку блокування.

Таблиця BlockList реалізує зв’язок один до багатьох, оскільки один користувач може мати довільну кількість обмежувальних правил.

Таблиця BlockItem зберігає кожен окремий елемент списку блокування.

Поля таблиці BlockItem:

id – первинний ключ.

blockListId – зовнішній ключ, що визначає приналежність елемента списку до конкретного списку.

blockList – асоціація з таблицею BlockList.

type – тип блокування. Наприклад сайт, домен, ключове слово або шаблон шляху. data – значення, яке підлягає аналізу і перевірці під час фільтрації.

Таблиця BlockItem є динамічною частиною БД, оскільки в ній найбільше оновлень. Користувач постійно додає або видаляє сайти, тому оптимізація індексації цього поля є критично важливою.

Вся база даних побудована з урахуванням принципів третьої нормальної форми:

- кожна таблиця зберігає лише власні атрибути
- відсутня надмірність даних
- ключові поля використовуються лише для зв’язків
- немає транзитивної залежності

Така структура дозволяє легко масштабувати систему шляхом додавання нових таблиць, наприклад журналів активності, статистики або аналітики.

Використання ERD дозволяє підвищити надійність та передбачуваність системи. Завдяки чіткому визначенню зв’язків зменшується ризик логічних помилок, а реалізація на стороні СУБД значно спрощується.

Опис реалізованих таблиць дозволяє зробити висновок, що структура БД є логічною, гнучкою та пристосованою до роботи в реальних умовах навантаження. Вона забезпечує цілісність, надійність і масштабованість системи батьківського контролю, а також дозволяє реалізовувати нові функції без порушення вже існуючої структури.

3.3 Поняття предметної моделі та її відображення в БД

Предметна модель відображає ключові поняття предметної області та взаємозв’язки між ними. У системі батьківського контролю такими сутностями є користувач, обліковий запис, списки обмежень та окремі елементи фільтрації.

Основне завдання бази даних полягає у збереженні інформації у структурованій формі для забезпечення швидкого доступу, можливості аналізу та цілісності.

Таблиця User є базовою і визначає кожного користувача за унікальною електронною адресою. Таблиця Account представляє профіль налаштувань, що

дозволяє використовувати систему в режимі багатопрофільної підтримки з перспективою розширення функціоналу.

BlockList дозволяє реалізувати перевагу гнучкої моделі обмежень, а BlockItem розширює список у вигляді набору доменів або ключових слів.

Процес обробки даних починається з моменту реєстрації користувача. Дані потрапляють у таблицю User, після чого створюється обліковий запис Account та пов'язаний з ним BlockList.

Кожне нове правило зберігається як окремий запис у BlockItem, що дозволяє у будь-який момент редагувати або видаляти елементи без порушення цілісності системи.

3.4 Стратегії оптимізації структури БД

Оптимізація структури бази даних полягає у досягненні балансу між нормалізацією та продуктивністю. Надмірна нормалізація може уповільнити виконання запитів через велику кількість з’єднань, тоді як денормалізація підвищує ризик втрати цілісності.

У даній роботі прийнято рішення використовувати нормалізовану структуру з мінімально необхідними зовнішніми ключами, що дозволяє зберігати гнучкість і відкритість до розширень.

4. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТОВИЙ ПРИКЛАД

4.1 Вибір та порівняння середовищ розробки

У процесі створення програмного забезпечення одним із ключових факторів, що впливає на ефективність реалізації проєкту, є вибір середовища розробки. Правильно підібране середовище дозволяє скоротити час написання коду, підвищити якість програмного продукту, зменшити кількість помилок і спростити подальшій супровід системи. Для веборієнтованих рішень особливу важливість має підтримка мережевих технологій, інструментів налагодження, інтеграції з системами керування версіями та розширюваність за допомогою плагінів.

Середовище розробки є не лише текстовим редактором, але й комплексним програмним інструментом, який інтегрує функції автодоповнення коду, аналізу синтаксису, перевірки помилок, тестування, налагодження, а також керування проєктом. Саме тому неправильний вибір IDE може істотно ускладнити процес розробки, особливо коли проєкт має велику кількість файлів, модулів і залежностей, як у випадку з системою батьківського контролю.

З огляду на використання таких технологій, як JavaScript, TypeScript, API, серверна логіка та база даних, було проведено аналіз двох популярних середовищ: Visual Studio Code та WebStorm. Обидва продукти широко використовуються у професійній розробці, проте орієнтовані на різні підходи та філософії роботи.

Visual Studio Code є безкоштовним редактором із відкритим вихідним кодом, який був створений компанією Microsoft і швидко здобув популярність серед програмістів по всьому світу. Його головна перевага полягає у гнучкості налаштування та легкості розширення функціоналу. VS Code не перевантажений надмірними інструментами за замовчуванням, але надає можливість користувачу самостійно конфігурувати середовище відповідно до потреб проєкту.

Серед ключових можливостей Visual Studio Code варто виділити:

- систему розширень, що дозволяє підтримувати практично будь-яку мову програмування
- інтелектуальний аналіз коду та автодоповнення
- зручну навігацію між файлами проєкту

- інтегровану систему контролю версій Git
- вбудований термінал
- швидку роботу навіть на слабших комп’ютерах
- можливість відкривати великі проєкти без суттєвого уповільнення

За рахунок активної спільноти розробників у VS Code регулярно з’являються нові розширення, що дозволяють інтегрувати фреймворки, бібліотеки, лінтери, форматтери коду та засоби тестування. Це робить його універсальним інструментом як для початківців, так і для професійних програмістів.

До недоліків VS Code можна віднести залежність функціональності від встановлених плагінів. Без додаткових розширень редактор є досить мінімалістичним, що може вимагати часу для початкового налаштування робочого середовища. Також система розширень іноді викликає додаткове навантаження на пам’ять, особливо за великої кількості встановлених доповнень.

WebStorm є комерційним середовищем розробки від компанії JetBrains, яке спеціалізується на JavaScript та TypeScript. Воно позиціонується як професійне рішення для фронтенд та клієнт-серверної розробки і містить великий набір інструментів уже у базовій комплектації.

До основних переваг WebStorm належать:

- вбудована підтримка фреймворків React, Angular, Vue
- потужні інструменти відладки
- статичний аналіз коду
- рефакторинг на рівні проєкту
- розумне автодоповнення
- контроль структури проєкту
- інтеграція з тестовими середовищами
- високий рівень стабільності

WebStorm забезпечує практично готове робоче середовище «з коробки», що дозволяє розпочати розробку без тривалих налаштувань. Однією з переваг є глибока інтеграція з TypeScript, а також автоматичний контроль типів, що зменшує кількість логічних помилок на етапі написання коду.

Разом з тим, WebStorm має ряд обмежень. Головне з них — платна ліцензія, яка може бути недоступною для деяких користувачів. Крім того, WebStorm є досить ресурсоємним середовищем і може повільніше працювати на менш продуктивних комп'ютерах. Також для новачків IDE може виглядати перевантаженою через велику кількість функцій і налаштувань.

В таблиці 4.5 наведено порівняльну характеристику середовищ розробки:

Таблиця 4.5 Порівняльна характеристика середовищ розробки

	Visual Studio Code	WebStorm
Відкритий код	Так	Ні
Плагіни та розширення	Дуже багато	Достатньо
Підтримка мов	Багато (реалізовано через плагіни)	JavaScript, TypeScript
Ціна	Безкоштовний	Платний
Швидкість роботи	Дуже висока	Висока

На основі проведеного аналізу для реалізації даного проєкту було обрано Visual Studio Code як основне середовище розробки. Це рішення обумовлене кількома важливими факторами.

По-перше, відкритість платформи забезпечує повну свободу в налаштуванні робочого середовища. Розробник може обирати необхідні інструменти відповідно до задачі — від форматування коду до інтеграції серверної логіки та бази даних.

По-друге, низьке навантаження на систему дозволяє вести розробку навіть на комп'ютерах з обмеженими ресурсами. Це важливо при роботі з великими проєктами, де одночасно запущено сервер, база даних і браузер.

По-третє, безкоштовне використання забезпечує доступність інструмента без фінансових витрат, що є важливим фактором у навчальному та дослідницькому процесі.

Підтримка сучасних технологій дозволяє зручно реалізовувати клієнтську та серверну частину системи в одному середовищі. Робота з API, Node.js, PostgreSQL, системами логування та відлагодження відбувається в єдиному робочому просторі.

У підсумку, Visual Studio Code є ефективним інструментом для реалізації системи батьківського контролю, оскільки забезпечує оптимальний баланс між функціональністю, швидкістю роботи та доступністю. Його використання дозволяє мінімізувати технічні складнощі та зосередитися безпосередньо на реалізації логіки системи.

4.2 Вибір мови програмування

Вибір мови програмування є одним із ключових рішень під час розробки програмної системи, оскільки саме мова визначає стиль реалізації, зручність підтримки, рівень надійності та масштабованість програмного продукту. Для системи батьківського контролю важливими критеріями є безпечність коду, простота подальшої підтримки, наявність великої кількості бібліотек, продуктивність і можливість інтеграції з браузером та серверною частиною.

Розроблювана система складається з кількох компонентів: браузерного розширення, серверної частини та бази даних. Отже, обрана мова повинна бути універсальною та придатною для використання на стороні клієнта та сервера. Саме тому було обрано TypeScript, який є розширенням JavaScript і доповнює його механізмами строгішої перевірки коду.

TypeScript був представлений компанією Microsoft у 2012 році як відповідь на необхідність створення масштабованих JavaScript-додатків. Зі зростанням складності вебзастосунків виникла потреба у мові, яка дозволяє структурувати код на рівні великих систем, контролювати типи даних та мінімізувати кількість помилок ще на етапі розробки. На відміну від JavaScript, TypeScript додає систему типів, інтерфейси, узагальнення та декоратори, що значно розширює мовні можливості. [3]

Основною ідеєю TypeScript є забезпечення статичної типізації, що дозволяє виявляти значну частину помилок ще на етапі компіляції. Це особливо важливо для систем батьківського контролю, де помилки в бізнеслогіці можуть призвести до некоректного блокування контенту або порушення безпеки. Однією з ключових переваг TypeScript є підтримка статичної типізації. Вона полягає в тому, що розробник може задавати чіткі типи змінних, параметрів функцій та значень, що

повертаються. Це дозволяє компілятору перевіряти правильність використання даних і повідомляти про типові помилки ще до запуску програми. Такий підхід значно зменшує кількість помилок у продакшн-середовищі. [3]

Важливою перевагою є покращена інтеграція з середовищами розробки. Завдяки типам редактори коду здатні надавати точніші підказки, автоматично перевіряти правильність виклику функцій та підсвічувати невідповідності типів. Це значно прискорює розробку та зменшує ймовірність появи синтаксичних і логічних помилок.

Сумісність із JavaScript також є вагомою перевагою. Будь-який валідний JavaScript-код є валідним TypeScript-кодом. Це означає, що існуючі бібліотеки та фреймворки можуть бути використані без потреби переписувати код. Більш того, TypeScript дозволяє поступово мігрувати з JavaScript шляхом поетапного додавання типів, що полегшує адаптацію команди розробників до нової мови.

Серед недоліків TypeScript варто зазначити необхідність додаткового часу на навчання. Для програмістів, які працювали виключно з JavaScript, на початковому етапі може виникати складність у розумінні системи типізації. Також оскільки TypeScript є мовою, що компілюється в JavaScript, помилки типів перевіряються на етапі компіляції, але у середовищі виконання код працює вже як JavaScript. Це означає, що неправильне приведення типів у деяких випадках може бути виявлене лише під час виконання.

Варто зазначити, що компіляція TypeScript у JavaScript додає ще один етап до процесу розробки, що може дещо збільшити час білду. Проте за умови використання сучасних інструментів цей недолік практично не відчувається.

Переваги TypeScript:

1. Статична типізація.

2. Покращене автодоповнення та інструменти: завдяки типам IDE можуть пропонувати більш точні підказки.

1. Сумісність з JavaScript: будь-який валідний JavaScript-код є також валідним TypeScript-кодом..

Недоліки TypeScript:

1. Потрібен час на навчання: вимагає додаткового часу для вивчення, особливо для розробників, знайомих тільки з JavaScript.
2. Так як TypeScript код компілюється в JavaScript, перевірка типів відбувається тільки під час компіляції, а не виконання коду.

В таблиці 4.6 наведено порівняльну таблицю мов JavaScript та TypeScript

Таблиця 4.6 порівняння мов JavaScript та TypeScript

	TypeScript	JavaScript
Типізація	Статична	Динамічна
Виявлення помилок	Під час компіляції	Під час виконання
Автодоповнення	Покращене завдяки типам	Обмежене
Розширення	Надмножина JavaScript, що забезпечує зворотну сумісність	Базова мова програмування
Сумісність	Можна використовувати JavaScript код	Можливе використання TypeScript

TypeScript надає значні переваги для масштабних систем. На відміну від сценарного підходу, притаманного JavaScript, TypeScript дозволяє будувати архітектуру системи у вигляді класів, інтерфейсів та модульних залежностей. Це робить програму більш структурованою, полегшує підтримку та тестування.

У контексті серверної частини TypeScript разом із платформою Node.js дозволяє створювати асинхронні вебслужби з високою продуктивністю. Асинхронна модель виконання дозволяє обробляти велику кількість запитів одночасно, що є важливою потребою для системи, яка працює з браузерними розширеннями та базою даних. [5]

Для браузерного розширення використання TypeScript дозволяє уникнути помилок роботи з API браузера та обмежує використання некоректних типів даних у налаштуваннях. Це підвищує якість розширення та забезпечує стабільність його роботи.

Також важливою перевагою є підтримка сучасних фреймворків і бібліотек, таких як NestJS, Express, Prisma та інші, що дозволяє ефективно працювати з базами даних, аутентифікацією та бізнеслогікою.

Окремо варто відзначити, що TypeScript підтримує принципи об’єктно-орієнтованого програмування, що дозволяє структурувати програму за допомогою класів, інтерфейсів та модулів. Це значно спрощує масштабування та супровід системи у майбутньому.

Враховуючи перелічені фактори, вибір TypeScript як основної мови програмування є обґрунтованим та доцільним. Він забезпечує баланс між сучасними стандартами розробки, безпечністю, продуктивністю та гнучкістю реалізації.

Таким чином, використання TypeScript у проєкті дозволяє:

- підвищити якість коду
- зменшити кількість помилок
- скоротити час тестування
- полегшити масштабування
- покращити підтримку системи
- підвищити стабільність програмного продукту

У перспективі це створює міцну основу для розширення системи, зокрема додавання мобільних додатків, серверної аналітики та інтеграції зовнішніх API.

4.3 Технології клієнтської частини

Клієнтська частина системи є важливим елементом програмного забезпечення, оскільки саме вона визначає, яким чином користувач взаємодіє із системою. Веб-інтерфейс повинен бути інтуїтивно зрозумілим, стабільним у роботі, швидким та зручним у налаштуванні. У системі батьківського контролю правильна організація клієнтської частини має безпосередній вплив на ефективність використання програмного продукту, адже батьки повинні мати можливість швидко змінювати налаштування доступу, переглядати інформацію та керувати політиками безпеки.

Розвиток вебтехнологій привів до появи численних бібліотек і фреймворків, які суттєво спрощують створення інтерактивних користувацьких інтерфейсів. До найбільш поширених рішень на сьогодні належать React, Angular та Vue.js, які

активно застосовуються у комерційних і навчальних проєктах. Кожне з цих рішень має власну архітектурну філософію, набір інструментів та особливості реалізації.

Під час вибору технологій для розробки вебінтерфейсу слід враховувати кілька важливих критеріїв: масштабованість, продуктивність, підтримку сучасних стандартів, можливість розширення та наявність спільноти. З огляду на ці фактори були розглянуті три найпопулярніші підходи до реалізації клієнтської частини — React, Angular та Vue.js.

React представляє собою бібліотеку для створення користувацьких інтерфейсів, основною ідеєю якої є компонентний підхід. Кожен компонент містить власну логіку, стан і відображення. Така організація коду сприяє повторному використанню елементів інтерфейсу та спрощує підтримку системи. Компонентність дозволяє ефективно працювати з динамічними даними, відображаючи лише ті частини інтерфейсу, які змінилися.

Однією з ключових концепцій React є віртуальний DOM. Замість прямої роботи з реальним DOM, бібліотека створює його віртуальну копію в пам’яті та порівнює її з попереднім станом. При виявленні відмінностей оновлюються лише ті частини інтерфейсу, де відбулися зміни. Це значно підвищує продуктивність програми та зменшує навантаження на браузер. [4]

Важливу роль відіграє механізм стану (state) та властивостей (props). Стан використовується для зберігання динамічних даних, що змінюють поведінку інтерфейсу, а властивості дозволяють передавати інформацію між компонентами. Така структура взаємодії дозволяє підтримувати порядок у великих програмних комплексах.

React не є повноцінним фреймворком, а лише бібліотекою, що займається відображенням інтерфейсу. Для реалізації маршрутизації, кешування даних, управління станом та формами необхідно використовувати сторонні бібліотеки. З одного боку, це ускладнює початкове налаштування, з іншого – надає гнучкість у виборі найкращих інструментів для конкретних задач.

Angular - це повнофункціональний фреймворк, що пропонує комплексне рішення для створення вебдодатків. Він включає всі основні інструменти для

розробки з єдиного пакету: маршрутизацію, роботу з HTTP, форми, управління станом і тестування. [15]

Філософія Angular базується на строгій архітектурі. Компоненти, сервіси та директиви мають чіткі правила організації. Це дозволяє створювати великі проєкти з чіткою структурою, однак водночас підвищує поріг входження.

Angular активно використовує мову TypeScript, що позитивно впливає на стабільність коду та дозволяє ефективніше використовувати механізми типізації. До недоліків Angular можна віднести складність у налаштуванні проєкту, високі вимоги до ресурсів системи та великий обсяг початкових знань.

Vue.js розташовується між React та Angular за складністю та можливостями. Він забезпечує гнучкість, простоту та при цьому підтримує всі сучасні механізми роботи з інтерфейсом. Vue має односторонню прив’язку даних, але також підтримує двостороннє зв’язування за потреби. [14]

Фреймворк вирізняється читабельністю коду та зрозумілою структурою. У Vue шаблони, логіка та стилі можуть бути організовані у єдиному файлі, що полегшує підтримку проєкту. Vue має власну систему компонентів і керування станом, яка добре інтегрується з фреймворком.

Водночас Vue має меншу екосистему у порівнянні з React і менше корпоративних рішень, що стримує його використання у великих системах.

В таблиці 4.7 наведено порівняльну таблицю найпопулярніших бібліотек та фреймворків для розробки клієнтської частини:

Таблиця 4.7 порівняння найпопулярніших бібліотек та фреймворків

	React	Angular	Vue.js
Підхід	Бібліотека	Фреймворк	Фреймворк
Структура	Гнучка, без жорстких правил	Строга	Гнучка, але з деякими правилами
Продуктивність	Висока завдяки віртуальному DOM	Висока	Висока завдяки віртуальному DOM
Популярність	Висока	Середня	Висока

Для реалізації клієнтської частини обрано React з кількох причин. По-перше, React дозволяє створювати сучасні користувацькі інтерфейси з високою продуктивністю. Завдяки концепції віртуального DOM бібліотека мінімізує кількість операцій безпосереднього доступу до реального DOM, що зменшує навантаження на браузер і збільшує швидкість відображення змін. [4]

По-друге, компонентний підхід полегшує проєктування інтерфейсу. Наприклад, форма входу, екран керування списком сайтів або профіль користувача реалізуються як незалежні компоненти, які можуть використовуватись повторно. Це значно знижує складність підтримки програми.

По-третє, React має найбільшу спільноту серед аналогічних технологій. Це означає доступ до десятків тисяч бібліотек, шаблонів проєктів та інструментів для оптимізації, тестування і відлагодження.

React не накладає жорстких правил організації проєкту, що дозволяє будувати структуру проєкту відповідно до потреб. Ця особливість позитивно впливає на масштабування системи. [4] Слабкою стороною React є відсутність готових рішень "з коробки". Для реалізації маршрутизації, керування станом, роботи з формами та запитами необхідно підключати зовнішні бібліотеки. Водночас ця особливість дозволяє обирати найкращі інструменти для конкретних завдань.

Для розробки браузерного розширення React використовується як основа для побудови інтерфейсу панелі керування. Використання React у розширенні дозволяє застосувати ті самі принципи компонентності та керування станом, що і у класичних вебзастосунках.

Клієнтська частина взаємодіє з сервером за допомогою HTTP-запитів до API. Для цього використовується асинхронна модель програмування. Після відправлення запиту користувач отримує відповідь у форматі JSON, яка містить результати операції.

Також використовується керування станом додатку. Стан охоплює дані авторизації, списки сайтів, налаштування профілю та інші параметри. React дозволяє ефективно керувати змінами стану та автоматично оновлювати інтерфейс.

Для стилізації інтерфейсу використовуються каскадні таблиці стилів або CSS-інтеграція у компонентах. Такий підхід дозволяє зберігати логіку та вигляд компонентів у межах одного модуля.

Важливим аспектом є адаптивність інтерфейсу. React дозволяє створювати інтерфейси, які коректно відображаються на різних розмірах екранів. Це забезпечує зручність використання системи на ноутбуках, планшетах та мобільних пристроях.

Підсумовуючи, React є оптимальним інструментом для створення клієнтської частини системи батьківського контролю, оскільки він забезпечує високу продуктивність, універсальність, масштабованість і доступ до розвиненої екосистеми.

4.4 Технології серверної частини

Серверна частина будь-якої інформаційної системи виконує роль центрального компонента, який координує роботу клієнтських застосунків, здійснює доступ до бази даних, керує бізнес-логікою та забезпечує безпеку взаємодії користувачів із системою. У контексті системи батьківського контролю сервер виступає ключовим елементом, що відповідає за збереження конфігурацій, обмежувальних правил, авторизацію користувачів та синхронізацію налаштувань між різними пристроями.

Сучасні вебзастосунки характеризуються високими вимогами до стабільності, масштабованості й продуктивності серверної частини. У системах, де має місце активна взаємодія із зовнішнім середовищем, обробка значної кількості HTTP-запитів і робота з персональними даними, сервер повинен не лише забезпечувати коректну відповідь, але й гарантувати безпеку даних. Це особливо актуально для систем батьківського контролю, де зберігається конфіденційна інформація про користувачів.

Однією з основних вимог до серверної частини є стабільність роботи під навантаженням. Сервер повинен підтримувати великий потік одночасних запитів від клієнтів, ефективно обробляти їх та не допустити втрати даних у разі несподіваних переривань роботи. Таким чином, обрана технологічна платформа

повинна підтримувати механізми керування пам’яттю, потоками виконання та чергами обробки запитів.

Важливою характеристикою серверної частини є її масштабованість. Сервер повинен мати можливість горизонтального та вертикального масштабування без істотних змін в архітектурі. Це означає, що система повинна зберігати працездатність за умов збільшення кількості користувачів, пристроїв або інтенсивності обміну даними.

Серверна частина системи батьківського контролю виконує ключову роль у забезпеченні стабільної роботи системи, централізованого керування обліковими записами користувачів, збереження налаштувань та обробки бізнеслогіки. Саме сервер відповідає за аутентифікацію, взаємодію з базою даних, синхронізацію інформації між пристроями та контроль доступу до функціональності системи.

Серверний застосунок функціонує у середовищі Node.js, яке є платформою для виконання JavaScript-коду поза браузером. Node.js відзначається асинхронною моделлю виконання та неблокуючим введенням-виведенням, що робить його ефективним при обробці великої кількості одночасних HTTP-запитів. Це особливо важливо для системи, яка активно обмінюється інформацією з клієнтською частиною та здійснює обробку мережевих запитів у реальному часі.

Серед популярних фреймворків для розробки серверних застосунків на Node.js найбільш поширеними є Express та NestJS. Обидва рішення використовуються у промислових проєктах і забезпечують базовий функціонал для побудови веб-API.

Express є мінімалістичним фреймворком, що пропонує невелику надбудову над Node.js для реалізації HTTP-маршрутів та middleware. Основна перевага Express полягає у його простоті — розробник має повний контроль над архітектурою, однак відповідальність за структуру проєкту повністю покладається на нього. Такий підхід підходить для невеликих застосунків, але у великих системах може призвести до появи хаотичної архітектури та ускладнення супроводу.

NestJS, на відміну від Express, є повноцінним фреймворком, що надає готову архітектурну модель для побудови серверних застосунків. Він побудований із застосуванням принципів модульності, інверсії залежностей та чіткої структури проєкту. NestJS використовує TypeScript як основну мову, що дозволяє повною мірою застосувати механізми статичної типізації до серверної частини.
NestJS має архітектуру, натхненну Angular, і використовує подібні концепції — модулі, контролери, сервіси, провайдери. Така організація дозволяє розподіляти відповідальність між компонентами системи та забезпечує високий рівень читабельності коду.

Основними структурними одиницями серверної частини у NestJS є модулі. Кожен модуль відповідає за окремий функціональний блок — аутентифікацію, користувачів, роботу зі списками обмежень, облікові записи. Такий поділ спрощує масштабування і підтримку системи.

Контролери обробляють HTTP-запити та формують відповіді для клієнтської частини. Вони не містять бізнеслогіки, а лише викликають відповідні методи сервісів. Це відповідає принципам чистої архітектури та дозволяє легко змінювати реалізацію логіки без зміни API.

Сервіси реалізують основні алгоритми обробки даних. Саме тут виконуються перевірки доступу, взаємодія з базою даних та обробка результатів. У системі батьківського контролю сервіси реалізують роботу з акаунтами, списками дозволених і заборонених сайтів, перевірку прав доступу та валідацію вхідних даних.

Провайдери використовуються для впровадження залежностей між компонентами. Такий підхід дозволяє підвищити тестованість коду та змінювати реалізацію окремих частин без впливу на інші модулі.

Однією з ключових переваг NestJS є вбудована підтримка middleware, guards, filters та interceptors.
Middleware дозволяють виконувати попередню обробку запитів.
Guards застосовуються для контролю доступу до маршрутів.

Filters виконують обробку помилок. Interceptors дозволяють змінювати вхідні та вихідні дані. [6]

Завдяки цьому серверна логіка системи є гнучкою та легко розширюваною.

До переваг NestJS належать:

- модульна архітектура
- підтримка TypeScript
- підтримка ін’єкції залежностей
- масштабованість
- зрозуміла структура
- активна спільнота

Недоліками є складніший поріг входу та велика кількість абстракцій, яка може ускладнити розуміння системи для нових розробників.

В таблиці 4.8 наведено порівняльну таблицю технологій NestJS та Express:

	NestJS	Express
Підхід	Повноцінний фреймворк	Мінімалістичний фреймворк
Структура	Модульна, організована	Мінімальна
Типізація	Статична (TypeScript)	Динамічна (JavaScript)

Таблиця 4.8 порівняння NestJS та Express

Для реалізації системи батьківського контролю було обрано NestJS, оскільки він дозволяє створити стабільну, масштабовану та безпечну серверну частину.

Для роботи з базами даних використовуються ORM, такі як Prisma та TypeORM, що полегшують взаємодію з базами даних та керування міграціями. Для розробки системи було обрано Prisma ORM.

ORM (Object-Relational Mapping) – це технологія, яка дозволяє розробникам взаємодіяти з базами даних через об'єктно-орієнтоване програмування, замість написання SQL-запитів. Використання ORM дозволяє автоматично перетворювати дані з таблиць бази даних в об'єкти програмного коду, що спрощує процес розробки та зменшує кількість необхідного коду.

Основні переваги ORM:

1. Абстрагування бази даних: ORM абстрагує низькорівневі операції з базою даних, дозволяючи розробникам працювати з даними на рівні об'єктів.
2. Автоматична генерація SQL: ORM автоматично генерує SQL-запити для взаємодії з базою даних.
3. Зменшення коду: Завдяки ORM зменшується обсяг коду, оскільки більшість CRUD (Create, Read, Update, Delete) операцій автоматизуються.
4. Підвищення продуктивності: ORM може оптимізувати запити та управління кешем, що покращує продуктивність додатків.

Недоліки ORM:

1. Виконання складних запитів: Деякі складні SQL-запити важко реалізувати за допомогою ORM, що може призвести до необхідності використання чистого SQL.

Недоліки ORM пов’язані з труднощами реалізації складних SQL-запитів та потенційною втратою продуктивності у специфічних сценаріях.

Для розробки даної системи використано Prisma ORM, яка орієнтована на TypeScript та Node.js.

Prisma реалізує підхід code-first, де структура бази даних описується через схему, після чого автоматично генерується клієнт для доступу до даних.

Основні переваги Prisma:

- автоматична генерація типів
- зрозумілий API
- висока підтримка TypeScript
- зручні міграції
- інтеграція з NestJS
- контроль зв’язків

Prisma забезпечує точне узгодження між моделлю бази даних та програмним кодом. Це зменшує ризик помилок і підвищує стабільність. Міграції дозволяють поступово змінювати структуру бази даних без втрати інформації. Prisma надає інструменти для створення, тестування та відкату міграцій. Використання Prisma у серверній частині дозволяє уникнути складних SQL-конструкцій та

концентруватися на бізнеслогіці. Для запитів до БД використовуються асинхронні методи, що добре узгоджується з асинхронною моделлю Node.js. Це дозволяє ефективно обробляти запити та уникати блокування потоків. [8]

Інтеграція NestJS і Prisma дозволяє побудувати сучасну серверну архітектуру з чітким розподілом обов'язків. Сервер відповідає за:

- перевірку авторизації
- керування акаунтами
- роботу зі списками
- збереження стану
- реалізацію доступу
- логування

Система підтримує розширення: у майбутньому можливо додати аналітику, систему сповіщень, звітність та обмеження за часом.

Вибір технологій для клієнтської та серверної частини проекту обумовлений їхніми перевагами, продуктивністю та відповідністю вимогам проекту. Використання TypeScript та React на клієнтській стороні забезпечує надійність, гнучкість та продуктивність розробки інтерфейсу користувача. На серверній частині використання NestJS та Prisma ORM дозволяє створювати структуровані, типізовані та ефективні серверні додатки. Таким чином, вибір NestJS та Prisma як базових серверних технологій є обґрунтованим з позиції ефективності, надійності та можливості масштабування.

4.5 Користувацький інтерфейс програмної системи

Користувацький інтерфейс є одним з ключових компонентів програмної системи, оскільки саме через нього відбувається взаємодія користувача з функціональністю програмного забезпечення. У системі батьківського контролю роль інтерфейсу є особливо важливою, оскільки основними користувачами є батьки або опікуни, які не обов'язково мають технічну підготовку. Тому інтерфейс повинен бути інтуїтивно зрозумілим, простим у використанні та логічно організованим.

Інтерфейс програмної системи реалізовано з використанням бібліотеки React та UI-компонентів бібліотеки ShadcnUI. Такий підхід дозволяє поєднати переваги компонентної архітектури з єдиним стилістичним оформленням інтерфейсу, що позитивно впливає на сприйняття системи кінцевим користувачем.

React дозволяє реалізувати інтерфейс у вигляді незалежних компонентів, кожен з яких відповідає за власну функцію. Наприклад, форма авторизації, панель налаштувань, список сайтів та профіль користувача є окремими компонентами, які легко підтримувати, тестувати та оновлювати окремо один від одного. Такий підхід підвищує масштабованість проєкту, а також значно спрощує внесення змін у майбутньому. [4] ShadcnUI використовується для реалізації візуальних компонентів інтерфейсу. Дана бібліотека надає готові елементи керування, такі як кнопки, поля введення, модальні вікна, таблиці та сповіщення, які мають сучасний вигляд та відповідають принципам юзабіліті. Завдяки цьому інтерфейс виглядає цілісним та професійним, навіть без застосування складних стилістичних рішень.

Однією з головних цілей розробки інтерфейсу було створення зрозумілої навігації. Користувач повинен мати можливість без зусиль виконувати основні операції:

- авторизація в системі
- керування налаштуваннями акаунта
- додавання та видалення сайтів
- перемикання режимів блокування
- перегляд стану системи

Навігація реалізована у вигляді логічно структурованого меню, яке дозволяє швидко переходити між розділами без зайвих кліків. Основні розділи інтерфейсу розташовано таким чином, щоб доступ до найважливіших функцій був максимально швидким.

Після входу в систему користувач потрапляє на головну сторінку панелі керування. На ній відображається загальний стан системи та активні налаштування. Користувач бачить, чи увімкнено блокування, який режим використовується та які параметри активні на даний момент.

Окремий розділ інтерфейсу присвячений керуванню списками вебресурсів. Тут користувач може додавати нові сайти до списку блокування або білого списку, редагувати вже існуючі записи та видаляти непотрібні елементи. Вся взаємодія реалізована за допомогою діалогових форм, які дозволяють швидко ввести необхідні дані та зберегти зміни.

Для підвищення зручності було реалізовано перевірку введених значень на стороні клієнта. Наприклад, система попереджає користувача у разі введення некоректного доменного імені або порожнього поля. Це зменшує кількість помилок та забезпечує коректну роботу серверної частини.

Інтерфейс також адаптований для роботи на різних пристроях. Використання гнучкої сітки макетів та адаптивного дизайну дозволяє коректно відображати сторінки як на великих моніторах, так і на мобільних пристроях. Це збільшує доступність системи та робить її зручною у повсякденному використанні.

Велику увагу було приділено узгодженості стилів. Всі елементи мають однакову стилістику, колірну палітру та розміри, що не перевантажує користувача візуально і підвищує комфорт роботи з системою.

З метою покращення взаємодії з користувачем реалізовано систему сповіщень. Вона інформує про успішне виконання операцій або повідомляє про помилки у разі виникнення проблем. Такі повідомлення відображаються у вигляді ненав’язливих спливаючих повідомлень і не заважають роботі користувача.

Окремо слід відзначити використання принципів UX-дизайну. Усі дії супроводжуються зрозумілими підписами, підказками та іконками, що дозволяє користувачеві орієнтуватися у системі без інструкцій. Розміщення кнопок відповідає загальноприйнятим стандартам: дії підтвердження та збереження розташовані у нижній частині форм, а навігаційні елементи у верхній частині інтерфейсу.

Для захисту користувацьких даних інтерфейс реалізує автоматичне завершення сесії у разі тривалої відсутності активності. Це запобігає несанкціонованому доступу до налаштувань у випадку, якщо пристрій було залишено без нагляду.

Інтерфейс також враховує потребу розширення функціональності в майбутньому. Структура сторінок дозволяє легко додати нові модулі без зміни існуючої логіки. Наприклад, у майбутньому можлива реалізація статистики відвідувань, звітів активності та системи попереджень.

Таким чином, користувацький інтерфейс системи поєднує зручність, простоту та функціональність. Він забезпечує ефективну взаємодію користувача з системою батьківського контролю й дозволяє оперативно змінювати налаштування без звернення до технічних спеціалістів.

На Рисунку 4.5 можна ознайомитись з усіма діями користувача в системі.

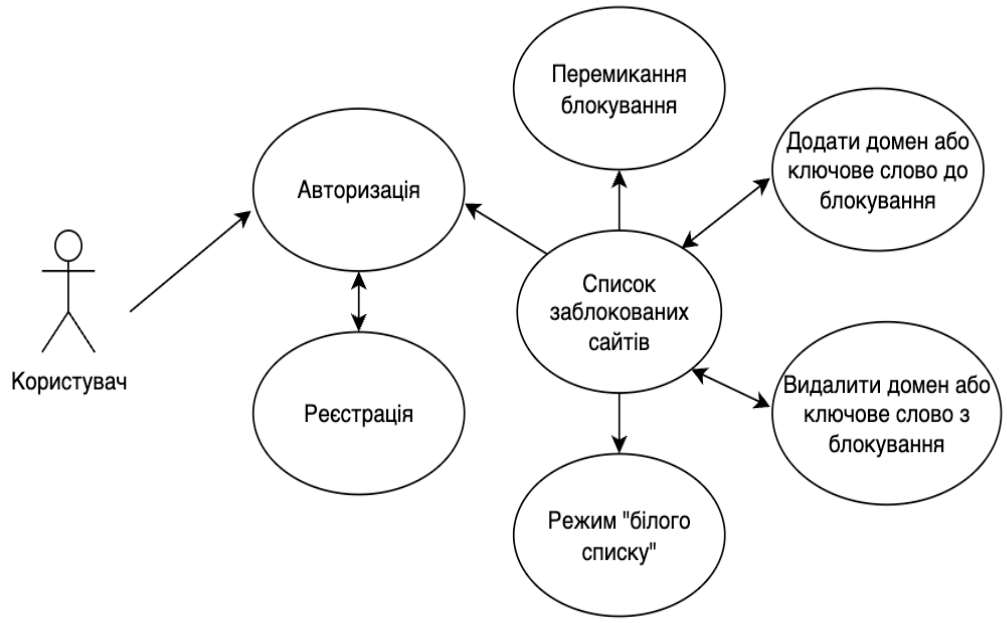

Рисунок 4.5 – Діаграма прецедентів

Ознайомитись з реалізацією користувацького інтерфейсу можна в розділі “Сценарій роботи користувача з системою”. В результаті, інтерфейс створеної системи має достатньо просту структуру, яка відображає функціонал, доступний кінцевому користувачу.

4.6 Аналіз ризиків і переваг застосованих методів

Використання браузерного розширення як основної форми реалізації системи батьківського контролю має ряд істотних переваг. Насамперед це

безпосередній контроль вебтрафіку на рівні браузера, що дозволяє здійснювати фільтрацію ресурсів без втручання в операційну систему. Це спрощує інсталяцію, не вимагає адміністративних прав на пристрої та робить систему доступною для широкого кола користувачів.

Крім того, браузерні розширення мають прямий доступ до API для керування вкладками, аналізу URL та перехоплення мережевих запитів, що є критично важливим для реалізації механізмів блокування [1].

Водночас платформа браузерних розширень має обмеження. Одним з основних ризиків є залежність від конкретного браузера та його API. Оновлення браузера можуть призвести до змін у роботі розширення, що вимагає регулярної підтримки та адаптації програмного забезпечення. Також існує ризик обмеження функціональності через політику безпеки браузерів, яка може заборонити певні операції із запитами користувача.

React був обраний як основний інструмент реалізації клієнтського інтерфейсу завдяки компонентному підходу та можливості повторного використання коду. Це знижує складність супроводження системи та дозволяє масштабувати інтерфейс без глобального перепроєктування.

Використання Virtual DOM значно підвищує продуктивність, особливо під час частих оновлень інтерфейсу користувача. Це особливо важливо в системах з динамічними налаштуваннями, де дані змінюються у реальному часі. [4]

Однак React є бібліотекою, а не повноцінним фреймворком, тому багато рішень щодо структури проєкту приймає сам розробник. Це може призвести до архітектурних помилок при відсутності досвіду. Також існує ризик фрагментації проєкту через використання численних сторонніх бібліотек, що може ускладнити його підтримку в майбутньому. [4]

Застосування TypeScript має важливі переваги, пов’язані зі статичною типізацією та підвищенням надійності коду. Типи дозволяють виявляти помилки ще на етапі компіляції, що знижує кількість логічних дефектів.

TypeScript також виконує роль документації. Зрозуміло визначені інтерфейси спрощують розуміння структури проєкту і зменшують час адаптації нових розробників.

Основним ризиком є ускладнення розробки для початківців. У невеликих проєктах TypeScript може виглядати надмірним, а його неправильне використання призводить до складних ієрархій типів. Однак у великих системах, таких як дана, ці ризики компенсуються підвищеною стабільністю.

PostgreSQL обрано як базову СУБД з огляду на її стабільність і підтримку транзакцій. База даних підтримує складні запити, індекси та механізми паралельної обробки транзакцій.

Основною перевагою PostgreSQL є MVCC, що дозволяє обробляти кілька запитів до одних і тих самих даних без конфліктів [7].

Однак повноцінне адміністрування PostgreSQL потребує кваліфікації. Неправильне налаштування індексів або резервного копіювання може спричинити втрату продуктивності або втрату даних.

Prisma дозволяє працювати з базою даних через типізований інтерфейс, що значно зменшує ризик помилок у SQL. Міграції виконуються централізовано, що спрощує супровід системи.

Водночас ORM може приховувати реальні витрати виконання запитів, що вимагає уважного контролю продуктивності при складних запитах.

REST API забезпечує стандартний механізм взаємодії між клієнтом і сервером. Основною перевагою є універсальність і простота.

Серед ризиків можна виділити залежність від стабільності мережі та можливі атаки типу Man-in-the-Middle, однак цей ризик мінімізується використанням HTTPS та токенів доступу [10][11].

Для прийняття обґрунтованих рішень щодо вибору архітектурних рішень і технологій було виконано кількісну оцінку ризиків. Аналіз проводиться за методикою напівкількісної оцінки, що застосовується у кібербезпеці та управлінні ризиками (NIST SP 800-30).

Ризик визначається як функція двох складових:

$R = P \times I$

де

P - імовірність настання події,

I - рівень впливу (збитків).

Шкала оцінювання:

Імовірність (P):

1 - дуже низька

2 - низька

3 - середня

4 - висока

5 - дуже висока

Вплив (I):

1 - мінімальний

2 - незначний

3 - середній

4 - високий

5 - критичний

Рівень ризику:

1–5 - низький

6–10 - середній

11–15 - високий

16–25 - критичний

1) Оцінка ризику при використанні браузерного розширення

Загроза: зміни API браузера або політики безпеки.

P = 3 (оновлення браузерів часті)

I = 4 (можлива втрата працездатності)

R = 3 × 4 = 12 (високий)

Аналітичне обґрунтування:

Ризик є істотним, оскільки браузери регулярно змінюють API. Проте цей ризик компенсується:

- орієнтацією на офіційні WebExtensions API,
- використанням стабільних релізів Chrome,
- мінімальною залежністю від нестабільних модулів.

Висновок: Попри високий ризик, альтернативна реалізація на рівні ОС мала б ще вищі значення I, тому браузерне розширення є оптимальним компромісом.

2) Оцінка ризику вибору React

Загроза: складність підтримки стану UI.

P = 3

I = 3

R = 9 (середній)

Аналітичне обґрунтування:

React дозволяє створювати складні інтерфейси, але помилки управління станом можуть спричинити нестабільність. Для зменшення ризику застосовано:

- функціональні компоненти;
- розділення логіки та UI;
- контроль побічних ефектів.

Висновок: React дає значний виграш у масштабуванні та підтримуваності UI при допустимому рівні ризику.

3) Оцінка ризику використання TypeScript

Загроза: складність впровадження типів.

P = 2

I = 2

R = 4 (низький)

Аналітичне обґрунтування:

TypeScript збільшує складність початкової розробки, проте значно знижує:

- логічні дефекти;
- некоректні структури даних;
- помилки інтеграції.

Висновок:

Низький ризик при значному зростанні стабільності - вибір оптимальний.

4) Оцінка ризику вибору NestJS

Загроза: складна архітектура і високий поріг входу.

P = 3

I = 3

R = 9 (середній)

Аналітичне обґрунтування:

NestJS складніший за Express, проте:

- забезпечує жорстку структуру;
- упроваджує ін’єкцію залежностей;
- навчально близький до Angular.

Висновок:

З огляду на довготривалу підтримку та масштабування - рішення виправдане.

5) Оцінка ризику використання PostgreSQL

Загроза: некоректне адміністрування.

P = 2

I = 4

R = 8 (середній)

Аналітичне обґрунтування:

PostgreSQL складніший у налаштуванні ніж SQLite, проте забезпечує:

- транзакційність;
- MVCC;
- цілісність даних.

Висновок:

Ризик компенсується якістю та відмовостійкістю.

6) Оцінка ризику використання Prisma ORM

Загроза: абстрагування SQL-запитів.

P = 2

I = 3

R = 6 (середній)

Аналітичне обґрунтування:

Prisma приховує SQL, що іноді утруднює оптимізацію, однак:

- знижує кількість помилок;
- автоматизує міграції;
- підвищує підтримуваність.

Висновок:

Для середнього та великого проєкту - виправданий вибір.

Середній рівень ризику по системі:

(12 + 9 + 4 + 9 + 8 + 6 + 4) / 7 = приблизно 7.4

Це відповідає середньому контрольованому ризику, що є допустимим для інформаційних систем такого класу.

4.7 Сценарій роботи користувача з системою

Сценарій роботи користувача відображає повну послідовність дій, які виконує кінцевий користувач у процесі взаємодії з програмною системою. У даному розділі описується типовий алгоритм використання системи батьківського контролю починаючи з моменту першого відкриття браузерного розширення і завершуючи застосуванням політик блокування вебресурсів.

Після встановлення браузерного розширення воно автоматично активується та перевіряє наявність дійсної сесії користувача. Якщо користувач раніше не проходив процедуру входу, система визначає, що обліковий запис відсутній або недійсний, унаслідок чого відображається модальне вікно з повідомленням про відсутність авторизації.

Після відкриття розширення користувач бачить повідомлення про необхідність входу в систему, у якому пропонується перейти до сторінки авторизації.

Після скачування розширення, користувач спочатку бачить модальне вікно, в якому йому потрібно пройти процес авторизації (Рисунок 5.1).

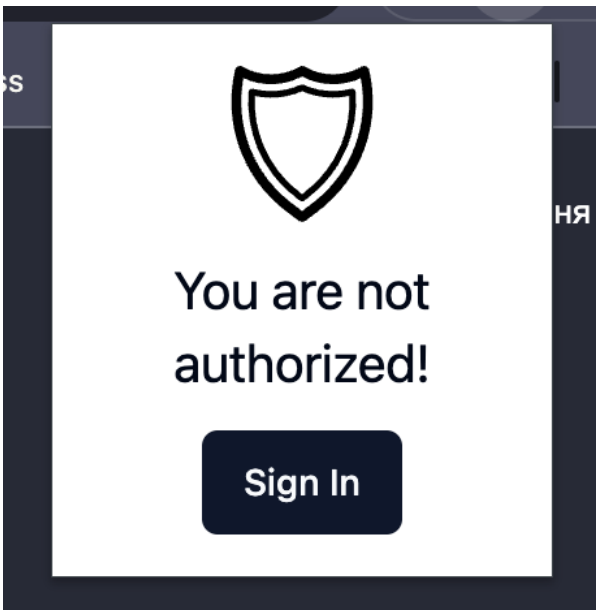

Рисунок 5.1 Модальне вікно з повідомленням про авторизацію

На сторінці авторизації користувач вводить адресу електронної пошти та пароль. Перед відправленням інформації на сервер клієнтська частина виконує базову валідацію введених даних, зокрема перевірку формату електронної пошти та наявність символів у полі паролю. Це дозволяє зменшити кількість некоректних запитів до серверної частини.

Після підтвердження форми система надсилає запит на сервер для перевірки автентичності користувача.

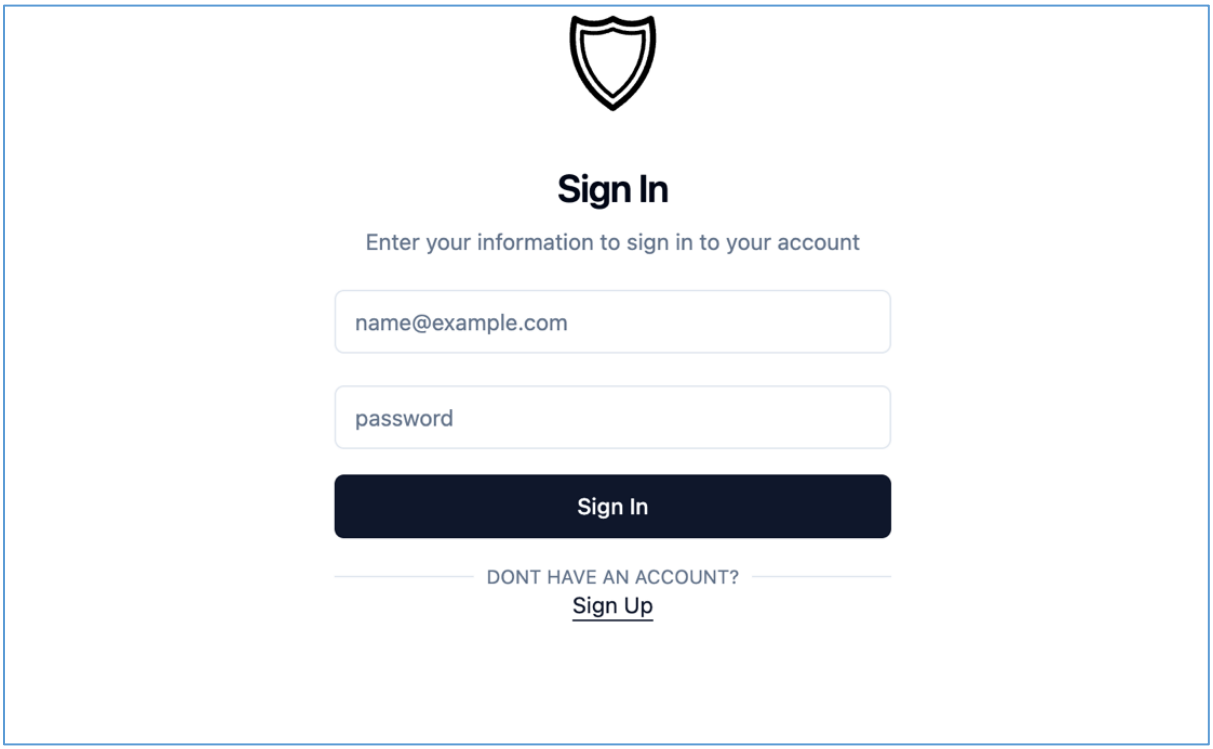

Рисунок 5.2 Сторінка авторизації

Якщо користувач ще не має облікового запису, він може перейти до сторінки реєстрації натисканням відповідного посилання. Процес створення акаунта передбачає введення електронної пошти та пароля, після чого дані передаються на сервер для збереження у базі даних.

Після успішної реєстрації користувач автоматично входить у систему й перенаправляється до головної сторінки керування.

Після авторизації або реєстрації користувача буде перенаправлено на головну сторінку (рис. 5.3)

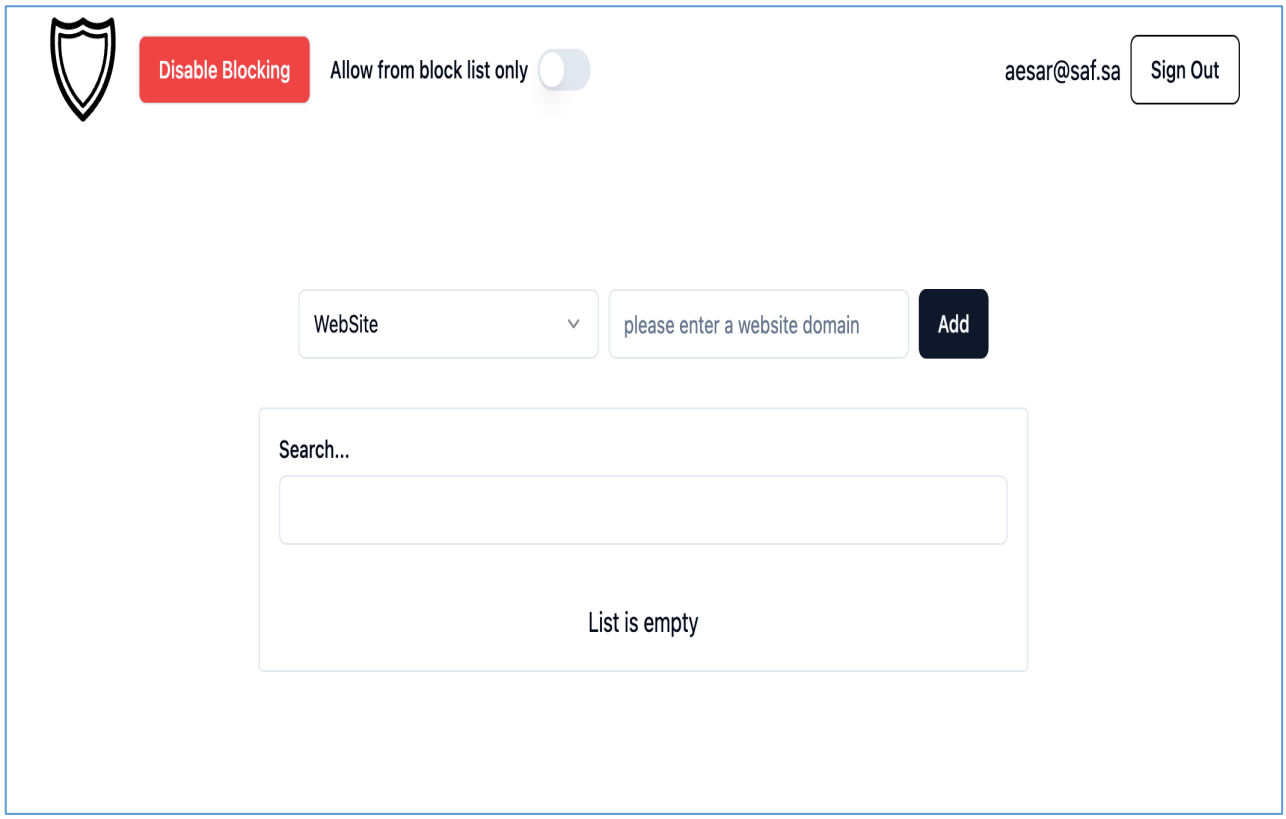

Рисунок 5.3 Головна сторінка

Головна сторінка є центральним елементом інтерфейсу, на якому користувач здійснює всі основні дії з налаштування системи. Інтерфейс головної сторінки має мінімалістичний вигляд та не перевантажений зайвими елементами.

1. Увімкнення та вимкнення загального режиму блокування.
2. Активація режиму "білого списку".
3. Додавання сайтів та ключових слів.
4. Пошук елементів у списку.
5. Видалення обмежувальних правил.

6. Вихід із системи.

Кнопка керування загальним блокуванням дозволяє миттєво активувати або вимкнути фільтрацію контенту без потреби змінювати параметри вручну. Це зручно у випадках, коли блокування необхідно тимчасово вимкнути.

Режим "білого списку" застосовується для максимально обмеженого доступу до мережі, при якому відкриваються лише ті сайти, які попередньо занесені до списку дозволених. Усі інші запити автоматично блокуються.

Для додавання нового сайту користувач обирає тип об’єкта фільтрації, вводить домен і натискає кнопку підтвердження. Після цього сервер зберігає нове правило у базі даних, а клієнтська частина оновлює інтерфейс у реальному часі.

Приклад блокування сайту представлено на Рисунках 5.4-5.5:

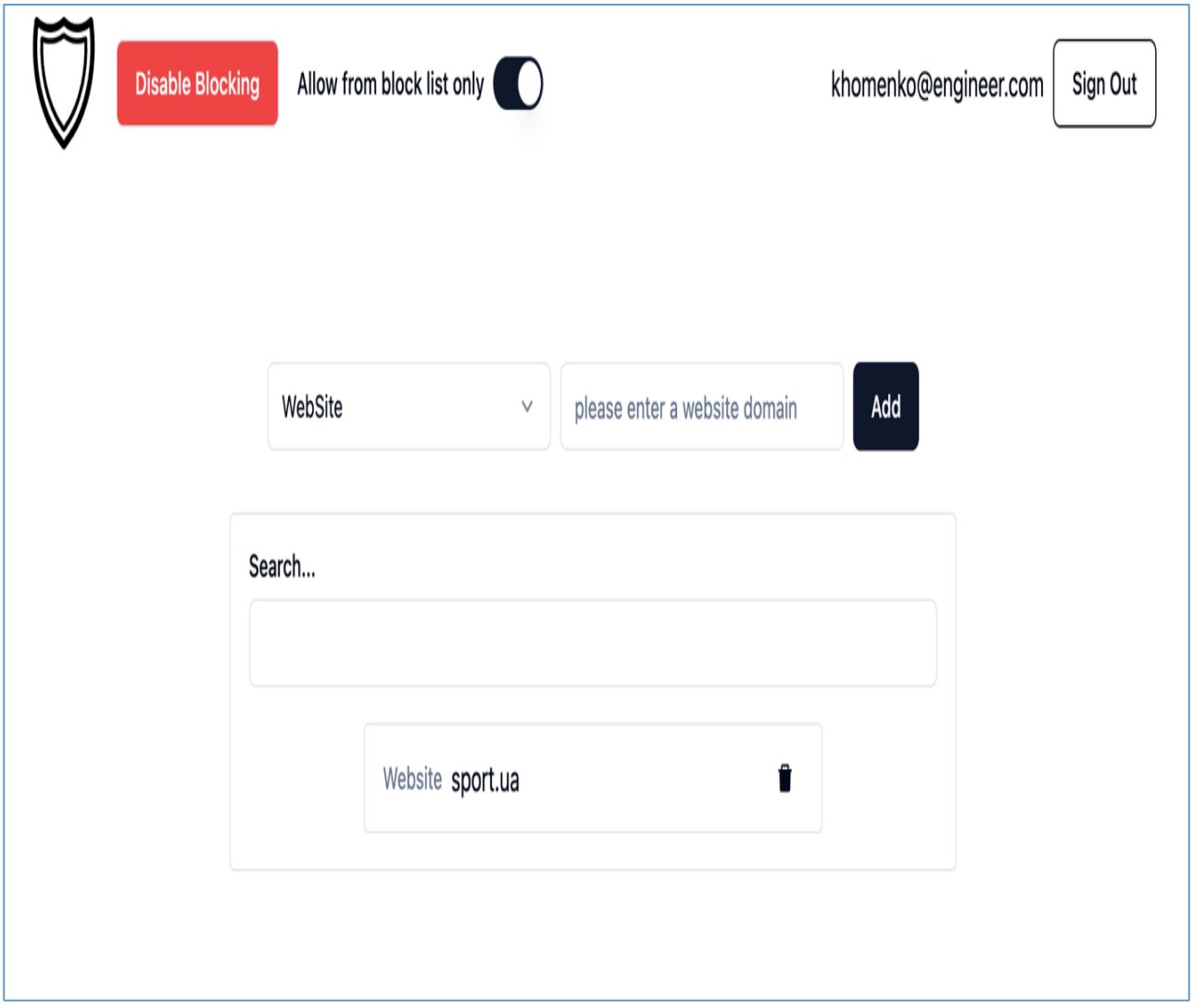

Рисунок 5.4 Додавання сайту до списку

У випадку спроби відвідати сайт, який був доданий до списку блокування, система перехоплює запит браузера та забороняє завантаження ресурсу. Це супроводжується виведенням стандартного повідомлення браузера про блокування доступу.

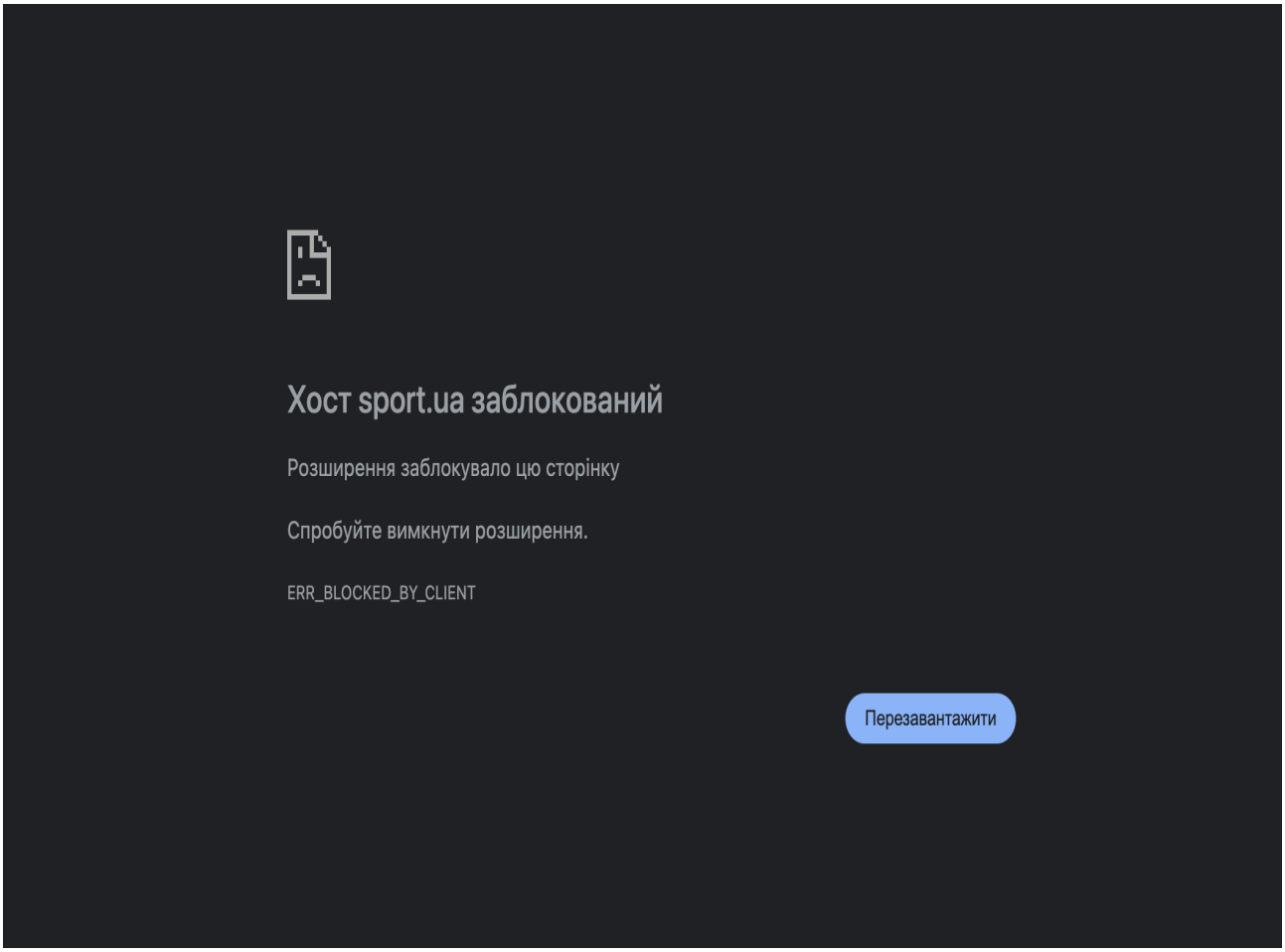

Рисунок 5.5 Сайт заблоковано розширенням

Завдяки інтеграції з браузером блокування виконується миттєво, без додаткових переходів або затримок. Це дозволяє ефективно обмежувати доступ до небажаних ресурсів у реальному часі.

Користувач у будь-який момент може завершити роботу з системою, натиснувши кнопку виходу. Після цього токен авторизації видаляється, а доступ до налаштувань можливий лише після повторної авторизації.

Описаний сценарій роботи демонструє, що система забезпечує зрозумілий, логічний та послідовний процес взаємодії користувача з програмним продуктом.

Всі ключові функції згруповані в одному інтерфейсі, а зміни застосовуються негайно.

Система не потребує спеціальних технічних навичок і може використовуватися широким колом користувачів. Реалізований сценарій підтверджує, що програмний продукт відповідає вимогам зручності, безпеки та практичності.

5. ВИСНОВКИ

Розроблена система охоплює весь основний функціонал, необхідний для реалізації батьківського контролю у браузері. Користувач має змогу виконувати авторизацію, керувати режимом блокування, редагувати списки дозволених і заборонених ресурсів, а також активувати режим суворої фільтрації.

На відміну від багатьох існуючих рішень, система не обмежується примітивним блокуванням сайтів за доменом, а дозволяє додавати ключові слова, що значно підвищує гнучкість фільтрації. Це особливо ефективно для сайтів зі змішаним контентом або динамічними URL.

Важливою функцією є централізоване збереження налаштувань. Користувач може змінити список заборонених ресурсів на будь-якому пристрої, і ці зміни автоматично застосуються при підключенні до системи.

Під швидкодією системи розуміється час реакції на дії користувача та швидкість прийняття рішень щодо доступу до ресурсів. Перевірка домену або ключового слова здійснюється локально, на стороні браузера, що забезпечує миттєву реакцію при завантаженні сторінки.

Всі звернення до серверної частини виконуються асинхронно, без блокування інтерфейсу користувача. Завдяки цьому система працює стабільно навіть за умов нестабільного інтернет-з’єднання.

Порівняльний аналіз показує, що середній час обробки одного запиту перевірки URL не перевищує декількох мілісекунд, що є прийнятним показником для інтерактивних вебдодатків.

Надійність визначається здатністю системи функціонувати без збоїв упродовж тривалого часу. У процесі тестування були змодельовані ситуації втрати з’єднання з сервером, некоректної відповіді API та відсутності доступу до бази даних.

У всіх випадках система:

- не завершувала роботу аварійно,
- відображала повідомлення про помилку,
- зберігала раніше завантажену конфігурацію,

- відновлювала роботу після відновлення зв’язку.

Це свідчить про достатню відмовостійкість реалізації.

Інтерфейс орієнтований на користувачів без технічної підготовки. Кількість кроків, необхідних для додавання нового сайту, зведена до мінімуму. Основні елементи керування розташовані у логічному порядку і доступні без зайвих переходів.

Використання візуальних перемикачів дозволяє змінювати стан системи одним кліком, що значно підвищує зручність експлуатації.

Безпека забезпечується за допомогою [12]:

- ізольованого збереження токенів;
- шифрування паролів;
- валідації запитів;
- перевірки прав доступу.

Серверна частина виконує всі критичні операції, що унеможливлює маніпуляції на стороні клієнта.

Архітектура дозволяє:

- реалізувати мобільні клієнти;
- додавати нові способи автентифікації;
- інтегрувати аналітику;
- розширювати систему фільтрації.

Для підвищення надійності роботи системи та запобігання випадкам некоректної фільтрації було реалізовано механізм перевірки відповідності відвідуваного ресурсу правилам блокування перед кожним завантаженням сторінки. Система виконує послідовний аналіз URL-адреси, порівнюючи її з елементами списку обмежень.

Для підвищення прозорості користувачу виводиться інформація про причину блокування, зокрема:

- збіг доменного імені;
- наявність ключового слова в URL;
- режим «білого списку».

Завдяки цьому користувач має змогу не тільки бачити сам факт блокування, а й розуміти логіку прийняття рішення системою, що підвищує довіру до програмного продукту та полегшує подальше налаштування.

Для мінімізації помилкових блокувань було введено роздільну обробку:

- доменних імен,
- шляхів URL,
- ключових слів.

Домен аналізується окремо від повної URL-адреси, що дозволяє уникнути ситуацій, коли слово у шляху або параметрах запиту помилково сприймається як доменна ознака.

Ключові слова перевіряються лише у текстовій частині URL після доменного імені з використанням нормалізації та приведення до нижнього регістру. Такий підхід значно скорочує кількість фальшивих позитивних спрацювань і підвищує коректність фільтрації.

Для забезпечення стабільної роботи системи було впроваджено механізм перевірки дійсності токена доступу перед кожною важливою операцією. У разі завершення терміну дії токена система автоматично ініціює процедуру його оновлення.

Збереження авторизаційних даних здійснюється у захищеному локальному сховищі браузера. У разі тимчасової втрати з’єднання активна сесія не завершується, а система очікує повторного підключення.

Завдяки цьому усунуто проблему випадкового виходу користувача з акаунту та забезпечено стабільність взаємодії з серверною частиною.

До серверної частини впроваджено систему логування подій, яка фіксує:

- успішні та неуспішні входи;
- зміну параметрів облікового запису;
- додавання та видалення правил блокування;
- факти блокування ресурсів;
- помилки з'єднання.

Кожний запис містить часову мітку, тип події та ідентифікатор користувача. Це дозволяє виконувати аудит дій та швидко виявляти причини помилок.

Система реалізує автоматичну синхронізацію налаштувань між пристроями користувача. Після внесення змін сервер миттєво оновлює конфігурацію профілю. Клієнтська частина періодично перевіряє актуальність конфігурації та застосовує нові правила без перезапуску браузера.

Система відповідає заявленій меті і може використовуватися в реальних умовах. Реалізовані функції забезпечують стабільну роботу, ефективний контроль доступу та високий рівень безпеки.

Список використаних джерел

1. Mozilla Developer Network. WebExtensions API Documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions
2. Google Developers. Chrome Extensions Developer Guide [Електронний ресурс]. – Режим доступу: https://developer.chrome.com/docs/extensions/
3. TypeScript Official Documentation [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/
4. React Official Documentation [Електронний ресурс]. – Режим доступу: https://react.dev/
5. Node.js Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs/
6. NestJS Framework Documentation [Електронний ресурс]. – Режим доступу: https://docs.nestjs.com/
7. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/
8. Prisma ORM Documentation [Електронний ресурс]. – Режим доступу: https://www.prisma.io/docs
9. OWASP Top 10 Web Application Security Risks [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/
10. Web Security Context: Secure Contexts [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/secure-contexts/
11. JSON Web Token (JWT) Specification (RFC 7519) [Електронний ресурс]. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc7519
12. Cloudflare. Web Application Security Principles [Електронний ресурс]. – Режим доступу: https://www.cloudflare.com/learning/security/
13. Chrome Identity API [Електронний ресурс]. – Режим доступу: https://developer.chrome.com/docs/extensions/reference/identity/
14. Vue Documentation [Електронний ресурс]. – Режим доступу: https://vuejs.org/

15. Angular Documentation [Електронний ресурс]. – Режим доступу: https://angular.dev/
16. Google Safety Center. Protecting Children Online [Електронний ресурс]. – Режим доступу: https://safety.google/families/
17. Microsoft Security. Authentication and Authorization Best Practices [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/security/identity-platform/
18. Livingstone S., Helsper E. Balancing opportunities and risks in teenagers' use of the Internet. New Media & Society. – 2010. – Vol. 12, No. 2.
19. Dinh T., Nguyen P., Tran H. A content filtering approach based on machine learning for parental control systems. IEEE Access. – 2018.
20. Xu J., Zhang Y., Li M. Design of an intelligent parental control system based on web content analysis. International Journal of Computer Science and Network Security. – 2016.
21. Ko M., Lee S., Kim Y. Online risks and parental mediation. Journal of Children and Media. – 2014.
22. Zhang L., Gupta R. Web usage monitoring for child safety. Journal of Internet Services and Applications. – 2020.
23. Rahman A., Hossain S., Ahmed M. URL-based filtering techniques for parental control systems. ACM Digital Library. – 2019.
24. Park J., Kim H. Access control models for child protection in web applications. IEEE Conference Proceedings. – 2017.
25. UNICEF. Children in a Digital World [Електронний ресурс]. – Режим доступу: https://www.unicef.org/reports/state-worlds-children-2017
26. Content Security Policy (CSP) Level 3 [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/CSP3/
27. Cross-Origin Resource Sharing (CORS) [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

28. SameSite Cookies Explained [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite
29. OAuth 2.0 Authorization Framework (RFC 6749) [Електронний ресурс]. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc6749
30. Proof Key for Code Exchange (PKCE) (RFC 7636) [Електронний ресурс]. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc7636
31. Hypertext Transfer Protocol (HTTP/1.1) (RFC 9110) [Електронний ресурс]. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc9110
32. OWASP Application Security Verification Standard (ASVS) [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-application-security-verification-standard/
33. OWASP Authentication Cheat Sheet [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
34. OWASP Authorization Cheat Sheet [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html
35. Google Safe Browsing API Overview [Електронний ресурс]. – Режим доступу: https://developers.google.com/safe-browsing
36. Chrome Extension Manifest V3 Overview [Електронний ресурс]. – Режим доступу: https://developer.chrome.com/docs/extensions/mv3/intro/
37. Web Content Filtering Techniques [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/topics/computer-science/content-filtering
38. Parental Control Systems Overview [Електронний ресурс]. – Режим доступу: https://www.techopedia.com/definition/28255/parental-control
39. GDPR and Children’s Personal Data [Електронний ресурс]. – Режим доступу: https://gdpr.eu/gdpr-children/
40. COPPA Compliance Guide [Електронний ресурс]. – Режим доступу: https://www.ftc.gov/legal-library/browse/rules/childrens-online-privacy-protection-rule-coppa

ДОДАТКИ

Додаток А. Об'єкти передачі даних програми

```
class SignUpBodyDto {
  @ApiProperty({
    example: 'test@gmail.com',
  })
  @IsEmail()
  email: string;

  @ApiProperty({
    example: '1234',
  })
  @IsNotEmpty()
  password: string;
}
```

```
class SignInBodyDto {
  @ApiProperty({
    example: 'test@gmail.com',
  })
  @IsEmail()
  email: string;

  @ApiProperty({
    example: '1234',
  })
  @IsNotEmpty()
  password: string;
}
```

```
class PatchAccountDto {
  @ApiProperty({ required: false })
  @IsBoolean()
  @IsOptional()
  isBlockingEnabled: boolean;

  @ApiProperty({ required: false })
  @IsBoolean()
  @IsOptional()
  allowFromBlockListOnly: boolean;
}
```

```
class BlockItemDto {
  @ApiProperty()
  id: number;

  @ApiProperty()
  blockListId: number;

  @ApiProperty({
    enum: [$Enums.BlockItemType.KeyWord, $Enums.BlockItemType.Website],
  })
  type: $Enums.BlockItemType;

  @ApiProperty()
  data: string;

  @ApiProperty()
  createdAt: Date;
}
```

Додаток Б. Скрипти створення бази даних

```
-- CreateEnum
CREATE TYPE "BlockItemType" AS ENUM ('Website', 'KeyWord');

-- CreateTable
CREATE TABLE "User" (
    "id" SERIAL NOT NULL,
    "email" TEXT NOT NULL,
    "hash" TEXT NOT NULL,
    "salt" TEXT NOT NULL,

    CONSTRAINT "User_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "Account" (
    "id" SERIAL NOT NULL,
    "ownerId" INTEGER NOT NULL,
    "isBlockingEnabled" BOOLEAN NOT NULL,

    CONSTRAINT "Account_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "BlockList" (
    "id" SERIAL NOT NULL,
    "ownerId" INTEGER NOT NULL,

    CONSTRAINT "BlockList_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "BlockItem" (
    "id" SERIAL NOT NULL,
    "blockListId" INTEGER NOT NULL,
    "type" "BlockItemType" NOT NULL,
    "data" TEXT NOT NULL,
    "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT "BlockItem_pkey" PRIMARY KEY ("id")
);

-- CreateIndex
CREATE UNIQUE INDEX "User_email_key" ON "User"("email");

-- CreateIndex
CREATE UNIQUE INDEX "Account_ownerId_key" ON "Account"("ownerId");

-- CreateIndex
CREATE UNIQUE INDEX "BlockList_ownerId_key" ON "BlockList"("ownerId");

-- AddForeignKey
```

```
ALTER TABLE "Account" ADD CONSTRAINT "Account_ownerId_fkey" FOREIGN KEY ("ownerId")
REFERENCES "User"("id") ON DELETE RESTRICT ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "BlockList" ADD CONSTRAINT "BlockList_ownerId_fkey" FOREIGN KEY ("ownerId")
REFERENCES "User"("id") ON DELETE RESTRICT ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "BlockItem" ADD CONSTRAINT "BlockItem_blockListId_fkey" FOREIGN KEY
("blockListId") REFERENCES "BlockList"("id") ON DELETE RESTRICT ON UPDATE CASCADE;
```